UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

APRENDIZAJE DIRIGIDO POR DETECCIÓN AUTOMÁTICA DE ERRORES DE PROGRAMACIÓN

PROYECTO FIN DE CARRERA
INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
ESPECIALIDAD EN SISTEMAS DE TELECOMUNICACIÓN

Autora: Ana González Ibáñez
Tutor: Valentín Moreno Pelayo
Julio 2015

Título: APRENDIZAJE DIRIGIDO POR DETECCIÓN AUTOMÁTICA DE ERRORES DE PROGRAMACIÓN

Autora: Ana González Ibáñez

Tutor: Valentín Moreno Pelayo

EL TRIBUNAL

Presidente:
José María Álvarez Rodríguez

Secretaria:
Manuela Alejandres Sánchez

Vocal:
José Luis de la Vara González

Realizado el acto de defensa del Proyecto Fin de Carrera el día 30 de Julio de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo: Presidente Fdo: Secretario Fdo: Vocal

AGRADECIMIENTOS

Al retomar mi vida universitaria no pude elegir un mejor tutor. Valentín todo lo veía fácil: -*Tardarás dos días en implementar el código*- me dijo. Gracias por tu optimismo, comprensión y empuje a la hora de abordar este proyecto.

Obviamente no han sido dos días, pero durante su realización y en este momento de mi vida, tanto profesional como académica, no puedo dejar de acordarme de otro proyecto, FinishersFinishers. En efecto, esta carrera se acaba.

No sé si esta etapa que finaliza era la prueba de natación, agobiante pero gratificante al ser la que más me gusta y la que más domino, donde el neopreno que me hace ir más rápido es mi familia y amigos.

También puede que ésta sea la etapa de bici, la de mayor belleza… No, esa no puede ser porque me caería mil veces… Aunque puede que concluyera también, gracias al apoyo que me ofrecerían mis compañeros de trabajo.

Quizás se trata de la maratón final, la de mayor desgaste, la más dura, en la que estás tú solo, con tu pasado, tu presente y tu futuro. Una etapa en la que la meta tiene una cinta y te dan la medalla al esfuerzo.

No sé en qué prueba me encuentro, pero ahora ya sé por qué dicen que, por duro que sea el recorrido, lo importante es llegar a la meta. Es así porque me enorgullece haber tenido en mi equipo aplaudiendo, aguantando, sufriendo, riendo, llorando a tantos animadores incondicionales.

Gracias a los flotadores: mamá, papá, mi mejor hermana Raki -a veces pienso que sólo tú me puedes entender, te quiero guapita-, Julitoo, tía, tío, Melin, Chimo, José, Clara, Jorge, Kino, Gloria, Miri, Ana, Nat, Vir, L, Cook, Dani, Laury, Bei, Lau, Maribeyy, Juanjo, Silvia –creadora de Finishers-, Alex, Rafa, mofa, porque cuando he necesitado aire habéis estado ahí.

Gracias a los ruedines: Sandriux -mi 5 en 1 cuando lo necesitaba, he contado contigo para engrasar la bici en todo momento, para tú, pupete-, Pau, La Porres, Gemuchi, Cogs, Martuki, Inesuki, Are… Porque ya no sé en qué equipo estáis pero me habéis dado el apoyo y la estabilidad que necesito para pedalear.

Gracias a mi avituallamiento, mi salvavidas, mis zapas, mi motor, mi pasado, mi presente, mi futuro, mi monito, mi finisher, mi 474, mi Aa Juan.

También quiero pediros perdón por llegar tan tarde a la meta. No penséis que será la última meta que cruce, porque van a venir muchas más, muchos retos y oportunidades.

Quizás sea el momento de hacer un Ironman.

finishers.

RESUMEN

La enseñanza de la programación es el objetivo de numerosas asignaturas; aunque los lenguajes sean diferentes, las reglas de estilo y los fines perseguidos son comunes a todas ellas.

La interacción personalizada entre profesorado y alumnado va a ser imprescindible, al depender de ésta el posterior éxito o fracaso del alumno. El problema es que fuera del entorno académico esa interacción no existe, o es difícil, si no imposible, que se produzca.

La realidad es que en los primeros instantes de acercamiento a la programación, el programador novel cuenta con la ayuda del compilador para corregir los errores de sintaxis de su código, pero a menos que tenga a un programador experto cerca, nadie le ayudará a corregir los errores de estilo.

Este proyecto de fin de carrera tiene como objetivo desarrollar una aplicación que evalúe si un programa cumple unas normas de estilo y proporcione al estudiante sugerencias o críticas constructivas. Con una herramienta de este tipo el alumno tendrá una mayor autonomía, recibiendo un valioso asesoramiento sobre todo en periodos en los que el contacto entre profesorado y alumnado es menor, o en los momentos de trabajo personal. Además, tendrá una mayor motivación ya que adquirirá de forma más eficiente las habilidades necesarias para la programación.

ABSTRACT

Teaching programming is the goal of many subjects; though the languages are different, the style rules and the objectives pursued are common to all of them.

The personal interaction between teachers and students will be imperative to rely on this a later success or failure of the student. The problem is that outside the academic environment that interaction does not exist or is difficult, if not impossible, to occur.

The reality is that in the first moments of approach to programming, the beginner programmer count on the help of compiler to correct syntax errors in the code, but unless you have an expert programmer close, nobody will help you correct the style errors.

This project aims to develop an application to assess whether a program meets standards of style and provide the student suggestions or constructive criticism. With such a tool, students will have more autonomy and received valuable advice especially in periods when the contact between teachers and students is lower, or in moments of personal work. You will also have greater motivation because it more efficiently acquire the skills needed for programming.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación del proyecto

La dificultad que encuentran los alumnos en su primer acercamiento a la programación queda patente en el gran índice de fracaso académico, en las titulaciones de Informática.

La capacitación en un segundo o posteriores lenguajes de programación y la problemática de su docencia, cuando el alumno ya cuenta con esas aptitudes del primero consolidadas, no presenta tantas dificultades, pero la interacción personalizada es primordial en el primer acercamiento a la programación [DET] .

El empleo de una herramienta automatizada puede ayudar a minimizar este problema, ya que una herramienta es un profesor incansable, siempre está disponible, no sanciona, es equitativo con todos los alumnos, en definitiva un ayudante en el que el estudiante va a confiar y al que puede acudir en cualquier momento.

Teniendo en cuenta que el 80% de los gastos en la industria informática está dedicado a mantenimiento de programas [UCE], nuestro segundo objetivo es acostumbrar a los programadores noveles a que documenten sus programas. Con frecuencia consideran una pérdida de tiempo insertar en el código un texto que no es en absoluto operativo y sólo lo hacen en el último momento, en aquellos ejercicios que deben entregar al profesor. Resulta adecuado tomar medidas para reducir este rechazo inicial, y conseguir que documenten y codifiquen simultáneamente, ya que esto facilita el posterior mantenimiento, el desarrollo del programa y el trabajo en equipo.

La integración en el entorno europeo exige la adopción de un enfoque educativo que priorice el aprendizaje del alumno respecto de la enseñanza del profesor. Esto significa que gran parte de la labor del profesor consiste en favorecer el autoaprendizaje del alumno. Sólo cuando hay esfuerzo por parte del alumno se produce verdadero aprendizaje .

El proceso de "enseñanza-aprendizaje" implica la interacción "profesor-alumno", entre otras razones, para aprovechar los posibles errores cometidos por este último como punto de partida de generación de nuevos aprendizajes, de un lado, y para la valoración del nivel de conocimiento alcanzado por otro. Es decir, el flujo de conocimiento no debe limitarse a ir del profesor al alumno, sino que este último debe aportar información al profesor para enriquecer el proceso educativo, mejorándolo dinámicamente en función de sus necesidades. Los errores y dudas que exprese el alumno servirán a ambos: al profesor para saber el grado de conocimiento del alumno y así poder reorientar su enseñanza, y al alumno para autoevaluarse
[MOR].

En este sentido, es factible desarrollar una aplicación que evalúe si un programa cumple

unas normas de estilo y proporcione al estudiante sugerencias o críticas constructivas. De hecho, Schrosch [SCH] desarrolló en 1995 una herramienta automática en esta línea de actuación. Sin embargo, como desde entonces se ha avanzado en el establecimiento de métricas que permitan la evaluación de proyectos software, pensamos que era un momento adecuado para construir otra que se adaptase mejor a las nuevas necesidades.

1.2. Objetivos

El objetivo fundamental de este proyecto es desarrollador una herramienta interactiva que contribuya a la formación de los programadores noveles (estudiantes). Dicha herramienta proporciona mayor autonomía a los estudiantes sobre todo en el tratamiento de los errores lógicos y de estilo complementando la información dada por el compilador. Se pretende que el nivel de conocimientos adquirido por los alumnos sea mayor y se consiga con más rapidez. En esta línea la aplicación facilita y orienta el aprendizaje fomentando el establecimiento de buenos hábitos de programación. Además, proporciona elementos de autoevaluación y en consecuencia, fomenta el uso de una metodología adecuada. Asimismo también se pretende mejorar el proceso de evaluación.

Utilizando esta herramienta el alumno tendrá una mayor autonomía, recibiendo un valioso asesoramiento, sobre todo en periodos en los que el contacto entre profesorado y alumnado es menor, o en los momentos de trabajo personal. Además tendrá una mayor motivación ya que adquirirá, de forma más eficiente, las habilidades necesarias para la programación.

La herramienta pretende servir de apoyo al alumno en la obtención de calidad del código durante el curso y durante todo el proceso de desarrollo de las aplicaciones de forma controlada y no únicamente al final, cuando tomar acciones correctivas resulta costoso y desmotivado [MIE].

El “Asesor de Programación” mejora los puntos débiles de las herramientas anteriores, entre otros, destacamos la mejora de la interfaz de usuario para que sea más amigable y la generación de documentación automática a partir de comentarios específicos embebidos en el código del alumno
[MOR].

La herramienta se ocupa de las asignaturas de Programación (Bilingüe, español) del Grado de Ingeniería Eléctrica y del Grado de Sistemas de Telecomunicaciones de la Universidad Carlos III de Madrid, consideradas de gran importancia en la comunidad universitaria internacional. En la actualidad, en las asignaturas seleccionadas, hay cuatro grupos de aproximadamente ochenta alumnos por grupo, lo que involucra a unos trescientos veinte estudiantes por asignatura, con un total de seiscientos cuarenta para las dos. En el proceso docente participan en torno a diez profesores, por lo que es importante la adecuada coordinación entre ellos.

Entre otras, este “Asesor de Programación” controla el cumplimiento de las normas de

estilo y errores lógicos detallados en los siguientes capítulos, para conseguir que los programas realizados por el alumno sean más legibles, reutilizables, mantenibles y robustos. En caso de que se observe que el programa codificado por el alumno, sintácticamente correcto en lenguaje Fortran o Java, no cumple alguna de estas normas de estilo, se le mostrará el mensaje correspondiente.

Además, se creará una documentación automática, para motivar al alumno en la fase de documentación, esperando que esta medida amortigüe el rechazo inicial que muestran los estudiantes a desarrollar esta tarea.

1.3. Estructura de la memoria

El presente documento se estructura en los capítulos siguientes:

Capítulo 1: INTRODUCCIÓN
Planteamiento del problema actual, descripción los fundamentos, motivaciones en los que se apoya un sistema de aprendizaje dirigido de programación, enumeración los objetivos principales y secundarios perseguidos con el presente proyecto y breve descripción de la estructura de la memoria.

Capítulo 2: ESTADO DEL ARTE
Se presentan los estudios actuales y conceptos sobre la detección de errores, donde se realiza un recorrido por los distintos errores de programación y, por otra parte, se ahondará en los diferentes tipos de enseñanza-aprendizaje actuales.

Capítulo 3: ANÁLISIS DE LA HERRAMIENTA
En este capítulo situamos este proyecto dentro del marco regulador de la calidad y de la innovación en la enseñanza superior universitaria. Por otra parte, se definen ciertos aspectos del sistema, como son: el alcance del proyecto, el entorno de desarrollo y recursos técnicos utilizados, los requisitos que cumple la herramienta y sus casos de uso.

Capítulo 4: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA
Se presenta el patrón la arquitectura empleada para el diseño y desarrollo del proyecto y su adopción a la medida de la herramienta desarrollada. Se describen las decisiones de implementación tomadas según los requisitos funcionales, y la interacción entre ellos.

Capítulo 5: CERTIFICACIÓN Y RESULTADOS
En este capítulo se presenta la estrategia de pruebas realizada para la validación de la herramienta y la verificación de los resultados obtenidos.

Capítulo 6: PLANIFICACIÓN Y PRESUPUESTO
Se muestra la planificación tanto teórica como real para llevar a cabo este proyecto, incluye un presupuesto del proyecto.

Capítulo 7: CONCLUSIONES Y TRABAJOS FUTUROS
Se presentan las conclusiones tras la realización del proyecto y el análisis de los resultados obtenidos. Además, se presentan posibles líneas de investigación en las que

seguir trabajando con el fin de obtener mejoras, tanto en la herramienta como en el aprendizaje del alumno.

Capítulo 8: REFERENCIAS Y BIBLIOGRAFÍA
Relación de material consultado y referenciado en la presente memoria.

Capítulo 9 : ANEXO
Incluye el Manual de Usuario de la herramienta además del glosario de términos utilizados en el presente documento.

2. ESTADO DEL ARTE

2.1. Introducción

Actualmente, cuando los conceptos de Web 2.0 y el e-Learning [ANG] , en los que la interacción con el usuario es el eje fundamental de las nuevas tecnologías y se han adentrado de lleno en los hogares y escuelas, surge una revolución educativa y tecnológica en el ámbito de la enseñanza de la programación, en la que se pretende fomentar y optimizar el potencial que presentan las técnicas de aprendizaje autodirigido.

Los entornos de desarrollos actuales cada día se vuelven más complejos, debido al aumento de las necesidades y cualificaciones del programador, por lo que el problema de enseñar a programar se vuelve fundamental para alcanzar, no solo el éxito de una determinada asignatura de programación, sino facilitar el aprendizaje de la programación a mediante un entorno de desarrollo profesional en el que el alumno continúe aprendiendo a medida que relaciona los mensajes de error con los motivos de dicho error [PER].

Los errores de programación (tanto en tiempo de compilación, como de ejecución) han sido siempre un reto dentro de la enseñanza de la programación. Esto se ve reflejado en las múltiples conferencias y publicaciones al respecto en los congresos de docencia del ámbito de la informática. Siendo un hándicap para los estudiantes a nivel nacional e internacional.

Los estudios sobre este tema se fundamentan en la relación existente entre los errores de programación, la falta de conocimientos teóricos y las malas prácticas de programación, y dan unas pautas a los docentes en las áreas y conceptos de programación que se tendrían que revisar y enfatizar en la enseñanza de la misma [FPP+] .

2.2. Análisis y detección de errores

2.2.1. Introducción

La comprensión de los errores es la clave para que el alumno aprenda de ellos, por lo que la corrección de los errores de un programa deben ser suficientemente clara para que el alumno entienda el mensaje de error, comprenda sus causas y proporcione la solución.

En primer lugar, los alumnos centran su aprendizaje en el funcionamiento correcto del programa más que en los aspectos de claridad y estilo, mantenibilidad y reusabilidad, tan importantes después a la hora de trabajar en equipo. Además, muchas veces, lo que ellos creen que es correcto no lo es. Simplemente se preocupan de que compile y aparentemente ejecute bien [MOR].

Lo que debemos plantearnos es si podemos crear un entorno de aprendizaje que guíe al alumno a prevenir y corregir errores de forma didáctica y efectiva.

Hay que aclarar que los errores en programación no son únicamente resultado de un desarrollador inexperto o un mal aprendizaje, si no que están a la orden del día, como Humphrey dijo:

"La fuente de muchos defectos software son simples descuidos y errores del programador "

[Humphrey, 1997]

2.2.2. Tipos de Errores de Programación

Los errores de programación se pueden clasificar en distintos tipos, dependiendo del momento en que se producen (antes o durante la ejecución) o de los efectos que ocasionan (si impiden la ejecución del programa o no) [RAN].

Tipos de errores según el momento en que se producen:

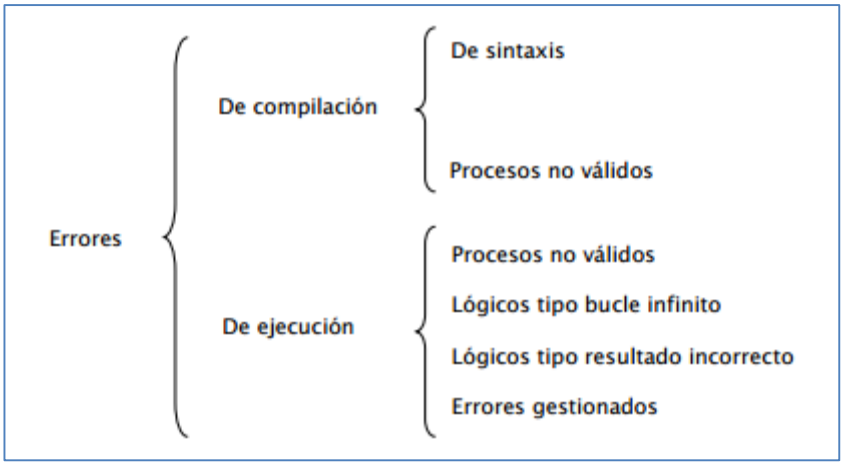


*Figura 1. Tipos de error según el momento de ocurrencia [**RAN**]*

Tipos de errores según los efectos que ocasionan:

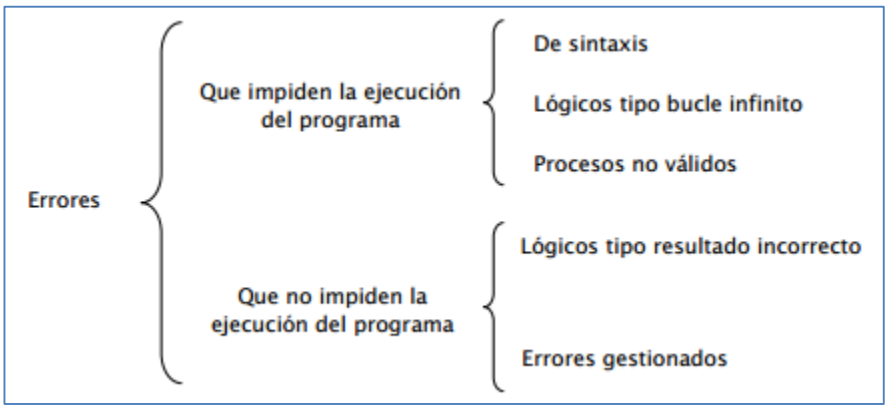


*Figura 2. Tipos de error los efectos que ocasionan [**RAN**]*

Errores de compilación

Son errores fácilmente detectables, pues impiden que el programa se ejecute. Cuando se ejecuta el programa, se compila el código en lenguaje binario, que el equipo entiende. Si el compilador no entiende alguna parte del código, alerta de un error de compilador.

Estos errores se deben a errores cometidos en la escritura del código. Los editores de programación identifican la mayoría de estos errores antes de ejecutar el código.

Errores en tiempo de ejecución

Son errores que se producen durante la ejecución del programa. Se producen cuando el programa intenta realizar una operación imposible de ejecutar. Por ejemplo, la división entre cero, este error sólo se reproduce en la ejecución, antes no es posible su detección.

Errores de Sintáxis

Estos errores se producen al infringir las normas de escritura de un lenguaje o por la omisión de términos o palabras obligatorias por los entornos de desarrollo y herramientas de corrección. Son fáciles de detectar y suelen deberse a descuidos o desconocimiento (programadores principiantes) y comprenden falta o mal uso de elementos separadores (uso de puntos, comas, etc.) o palabras mal escritas.

Procesos no válidos

Los procesos no válidos se producen por el propio programa o por causas ajenas al programa. Por ejemplo: que no se introduzca el dato esperado, que un archivo no contenga la cantidad de datos que se espera o que un dato o fichero no exista o no esté donde se espera. Son errores que normalmente se suelen detectar y gestionar gracias a las herramientas de corrección y depuración del entorno de desarrollo del lenguaje que utilicemos.

Algunas habituales son:

- Uso de variables no declaradas.
- Modificaciones no permitidas del número de localizadores de un array dinámico.
- Llamadas a módulos que no existen.
- Pasar un número incorrecto de parámetros o argumentos a un módulo, la ausencia de parámetros si el módulo los requiere.
- Asignar un valor a una variable que no coincide con el tipo de dato con el que se declaró dicha variable.
- Indeterminaciones matemáticas (por ejemplo: raíz cuadrada de un número negativo, división entre 0).
- Intentar leer datos o usar ficheros inexistentes [RAN].

Las dos últimas son difíciles de detectar si las variables dentro de la expresión hacen

llamadas a otros módulos y es ahí donde cambian su valor, serían errores durante la ejecución.

Errores lógicos

Son errores que generan resultados indeseados o inesperados e impiden que el programa obtenga el resultado esperado. Son los errores más difíciles de detectar y, por tanto, de corregir pues el programa compila y se ejecuta sin errores pero no responde a lo deseado [RAN].

Existen dos tipos de errores lógicos:

1. **Errores lógicos tipo bucle infinito**: estos errores hacen que la ejecución se bloquee, el programa se bloquea, sin obtener un informe de errores. Puede tratarse de procesos no infinitos pero que consumen tanto tiempo y recursos que exceden al máximo permitido.

2. **Errores lógicos tipo resultado incorrecto**: estos errores no bloquean el flujo del programa pero devuelven un resultado incorrecto. Dependiendo del nivel de desviación del error con respecto al resultado esperado, se puede facilitar su detección y corrección, cuanto más se desvíe del resultado, más fácil es su detección. Tras la ejecución del programa, el entorno de desarrollo no puede saber si el resultado es el esperado por el programador.

Errores gestionados

Un error es gestionado si, ante la presencia de uno de los anteriores errores, existe una previsión del error con un procedimiento a seguir, llamado gestión vía detección.

Al producirse este error, es detectado y, a continuación, se procede a su gestión. Lo que se pretende tanto en la enseñanza como en las herramientas disponibles para detección de errores es que la mayor parte de errores posibles sean detectados y gestionados, sin embargo esto no garantiza la calidad ni el buen funcionamiento del programa.

Estos errores dan una idea del control que el programador tienen sobre el flujo del programa con el fin de obtener solución del mayor número de requisitos posibles del sistema, bien sea parciales o reorientados a una resultado controlado.

Los errores lógicos de tipo resultado incorrecto no pueden tratarse como gestionados porque para el sistema el error no existe. La única posibilidad de gestionarlos es a su vez la vía lógica a través de estrategias de control de resultados, que puede resultar muy complejo [RAN].

2.2.3. Entornos de desarrollo actuales y sus limitaciones

Un entorno de desarrollo integrado (IDE), es un entorno de programación que ha sido empaquetado y que está compuesto por un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI), de manera que funciona como un

programa de aplicación.

Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Los IDE proveen de un escenario de trabajo para la mayoría de los lenguajes de programación tales como C++, Java, Fortran, etc. Un IDE puede servir para desarrollar en un solo lenguaje de programación o varios (Eclipse, Visual Basic, etc.) [GAR].

A pesar de ello, los entornos de desarrollo utilizados para la programación en un determinado lenguaje no siempre ayudan en la dinámica de aprendizaje de la programación, siendo difícil integrar los warnings y errores con los conceptos teóricos impartidos en la asignatura. Además, en ocasiones, la explicación de los errores es tan pobre para un programador sin experiencia que se hace compleja la gestión del error (ver Figura 3).

Los entornos de desarrollo no enseñan buenas prácticas de diseño, y carecen de guías que permitan al programador mejorar su estilo de programación.

"Los IDE son sistemas pasivos que analizan el código de forma superficial "

[Jacobson 2002]

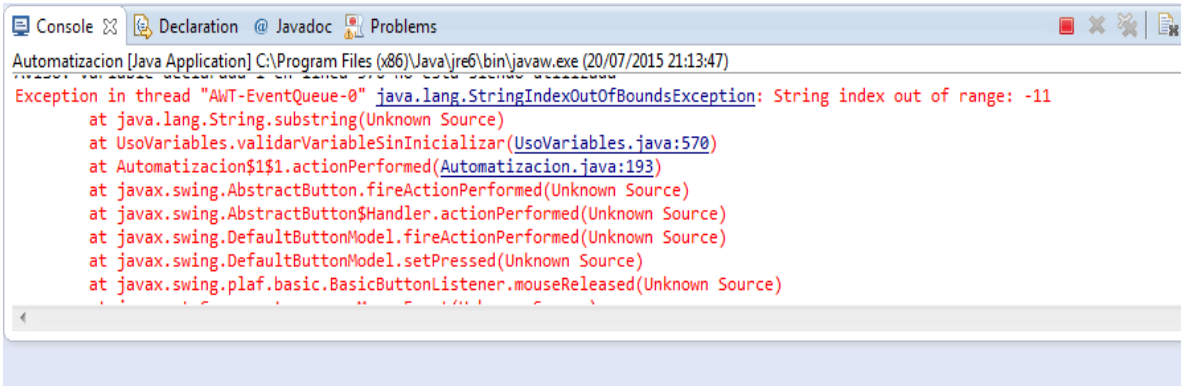


Figura 3. Ejemplo de error de un entorno de desarrollo integrado (Eclipse)

2.3. Aprendizaje de la programación

2.3.1. Introducción

En los últimos años, tanto el aprendizaje como la enseñanza de la programación han ido evolucionando, ha pasado de ser una disciplina altamente teórica y con niveles bajos de práctica, a la tener que adaptarse a un programa europeo Bolonia que instaura nuevas tecnologías docentes, en detrimento de los antiguos plantes de estudio, donde se empleaban clases magistrales.

Dentro de este marco educativo se fomenta la evaluación continua, donde se realiza un seguimiento diario del trabajo personal del alumno mediante evaluaciones periódicas.
Para llevar la evaluación continua se sugieren dos medios:

1. Uso de Internet y las nuevas tecnologías TIC, con todas las posibilidades que ofrece, y las tutorías personales.
2. Enseñanza práctica con la intervención activa del alumno a través de participación en clase, ejercicios, trabajos en grupo y prácticas profesionales en empresa.

Actualmente la mayoría de estudiantes tienen a su disposición un ordenador propio, por lo que la enseñanza de la programación tiene mayores posibilidades de aplicación práctica de los conceptos aprendidos en clase de manera teórica. Además los actuales entornos de desarrollo, como se ha mostrado, cada vez son más completos y ofrecen mejoras sustanciales en el uso de los lenguajes de programación, de manera que no hay que instalar varias herramientas, librerías adicionales o plugin´s a la hora del desarrollo de un proyecto.

Según el Real Decreto 1538/2006, Capítulo IV “Definición del currículo por las Administraciones educativas”, en su Artículo 18. “Adaptación al entorno socio-productivo”, Aptdo. 4, por el que se establece la ordenación general de la formación profesional del sistema educativo:

> *"La metodología didáctica de las enseñanzas de formación profesional integrará los aspectos científicos, tecnológicos y organizativos que en cada caso correspondan, con el fin de que el alumnado adquiera una visión global de los procesos productivos propios de la actividad profesional correspondiente”.*
>
> (BOE-A-2007-92)[BOE]

Como se puede intuir, el entorno docente es parte fundamental en la adopción de uno u otro método de estudio. Los estudiantes se adaptan al contexto e influencia del educador. Dicho contexto es decisivo en la estrategia que cada estudiante adopta en el proceso de aprendizaje, dicha estrategia surge del estudiante como una adaptación racional a las exigencias del mencionado para asegurarse así un buen resultado. Es más, la evaluación, entendiéndola como exámenes dentro un sistema de puntos, supedita al

alumno en el enfoque de su estrategia educativa
[MOR].

La manera en que se evalúa al estudiante afecta en lo que es importante para ellos a la hora de enfocar el estudio, pues es evidente que el objetivo de los alumnos es aprobar, por lo que se establecen unos medios para un fin.

Ante la asignatura de la programación y la masificación de información, el profesor tiene la responsabilidad principal de encaminar, ordenar y crear las condiciones para favorecer el estudio cognitivo de alto nivel, que incluya pensamiento crítico y auto-orientación.

> *"El desarrollo de software nunca ha sido tan complejo como lo es ahora. Los desarrolladores de software trabajan intensivamente con el conocimiento. No sólo deben comprender nuevas tendencias y tecnologías, sino que necesitan saber cómo aplicarlas de forma rápida y productiva".*
>
> [Ivar Jacobson, 2002]

2.3.2. Métodos y técnicas de enseñanza

Existen actualmente multitud de técnicas y métodos de enseñanza que en muchas ocasiones son empleados de manera empírica y aplicándose de manera incompleta, debido a veces al desconocimiento o a la falta de formación al respecto. Es de vital importancia analizar y conocer a través de la práctica los diferentes principios, teorías al respecto, así como los métodos desarrollados para conseguir un alto nivel educativo en los procesos formativos tanto del estudiante como del docente.

A continuación se describen los diferentes métodos y técnicas de enseñanza actuales.

Principios de Aprendizaje

Los principios de enseñanza y aprendizaje que se mencionan a continuación, fueron recopilados por la Universidad de Carnegie Mellon, responden a los alineamientos pedagógicos que giran en torno a [ROS]:

- Un modelo centrado en el estudiante: donde el protagonista es el alumno y el profesor es el mediador. Lo importante es el escenario que crea el profesor para que los estudiantes construyan su conocimiento.
- Aprendizaje activo: los estudiantes se involucran mediante sus propias experiencias de aprendizaje. Estas experiencias se desarrollan en colaboración.
- Aprendizaje significativo: el estudiante comprende lo aprendido y lo aplica a situaciones o contextos, conectando la información nueva con los conocimientos previos.

Según lo anterior, los 7 principios de aprendizaje son [MEL]:

1. **Conocimientos previos.** El conocimiento previo de los estudiantes puede ayudar o dificultar el aprendizaje:
   Los estudiantes aprenden al conectar nueva información con lo que ya conocen. Si el conocimiento previo de los estudiantes es robusto y precisa y se activa en el momento apropiado, proporciona una base sólida para la construcción de nuevos conocimientos. Aunque el conocimiento previo es la base para nuevos aprendizajes, puede interferir cuando el conocimiento es inerte, insuficiente para la tarea, que se activa de forma inapropiada, o inexacta, puede interferir o impedir un nuevo aprendizaje.

2. **Organización del conocimiento**. ¿Cómo los estudiantes organizan el conocimiento y aprenden y aplican lo que saben?:

   Los estudiantes de forma natural llegan a hacer conexiones entre piezas de conocimiento. Cuando esas conexiones se forman y el conocimiento se organiza con precisión y de manera significativa, los estudiantes son más capaces de recuperar y aplicar sus conocimientos de manera eficaz y eficiente. Por el contrario, cuando el conocimiento está conectado de manera inexacta o aleatoria, los estudiantes pueden dejar de recuperar o aplicarla adecuadamente.

3. **Motivación.** La motivación de los estudiantes determina, dirige y sostiene lo que hacen para aprender.

   Mientras los estudiantes entran en la universidad y obtienen una mayor autonomía sobre qué, cuándo y cómo estudiar y aprender, la motivación juega un papel fundamental en la orientación de la dirección, intensidad, persistencia, y la calidad de los comportamientos de aprendizaje en las que participen. Cuando los estudiantes encuentran un valor positivo en una meta o actividad de aprendizaje, y perciben el apoyo de su entorno, es probable que encuentren la motivación para aprender.

4. **Dominio.** Para desarrollar la maestría, los estudiantes deben adquirir habilidades de componentes, la práctica la integración de ellos, y saber cuándo hay que aplicar lo que han aprendido:

   Los estudiantes deben desarrollar no sólo las habilidades de los componentes y los conocimientos necesarios para realizar tareas complejas, también deben practicar combinando e integrando a desarrollar con una mayor fluidez y automaticidad. Por último, los estudiantes deben aprender cuándo y cómo aplicar las habilidades y conocimientos que aprenden. Como instructores, es importante que el profesor muestre los elementos de dominio con el fin de ayudar a nuestros estudiantes a aprender más efectivamente.

5. **Práctica y retroalimentación**. Práctica dirigida a objetivos junto con retroalimentación específica mejora de la calidad del aprendizaje de los estudiantes:

El aprendizaje y el rendimiento es mejor cuando se fomentaron los estudiantes participan en la práctica que se centra en un objetivo o criterio específico, dirigido a un nivel apropiado de dificultad, y se realiza con la frecuencia suficiente para satisfacer los criterios de rendimiento. La práctica debe ir acompañada de retroalimentación que se comunica explícitamente sobre algún aspecto del rendimiento de los estudiantes en relación con criterios específicos, proporciona información para ayudar a los estudiantes a progresar en el cumplimiento de esos criterios, y se da en un momento y la frecuencia que le permite ser útil.

6. **Desarrollo y clima del curso.** Nivel actual de desarrollo de los estudiantes interactúa con el clima social, emocional e intelectual del curso para impactar el aprendizaje:

   Los estudiantes no son sólo receptores de intelecto, sino también seres sociales y emocionales, y que todavía están desarrollando la gama de habilidades intelectuales, sociales y emocionales. Si bien no podemos controlar el proceso de desarrollo, podemos dar forma a los aspectos intelectuales, sociales, emocionales y físicas de clima en el aula de manera apropiadas para el desarrollo. De hecho, muchos estudios han demostrado que el clima que creamos tiene implicaciones para nuestros estudiantes. Un clima negativo puede impedir el aprendizaje y el rendimiento, pero un clima positivo puede dinamizar el aprendizaje de los estudiantes.

7. **Autodirigido.** Para convertirse en aprendices autodirigidos, los estudiantes deben aprender a controlar y ajustar sus enfoques de aprendizaje:

   Los estudiantes pueden participar en una variedad de procesos metacognitivos para monitorizar y controlar su aprendizaje-evaluación de la tarea en cuestión, la evaluación de sus propias fortalezas y debilidades, la planificación de su enfoque, la aplicación y el seguimiento de diversas estrategias, y reflexionar sobre el grado en que su enfoque actual es de trabajo. Por desgracia, los estudiantes tienden a no participar en estos procesos de forma natural. Cuando los estudiantes desarrollan las habilidades necesarias para participar de estos procesos, adquieren hábitos intelectuales que no sólo mejoran su rendimiento, sino también su eficacia como estudiantes.

Método y Técnica

El método de enseñanza es un conjunto de técnicas coordinadas para gestionar el aprendizaje del alumno a los objetivos establecidos, es por tanto, una guía para el docente, no es inmutable y debe tener como objetivo final la superación intelectual y el autoaprendizaje del alumnado [SAL].

Tipología de Métodos

Se clasifican teniendo en cuenta la forma de razonamiento, la coordinación de la asignatura, la disposición del alumno en cuanto a aspectos disciplinarios y de

organización escolar [SAL]:

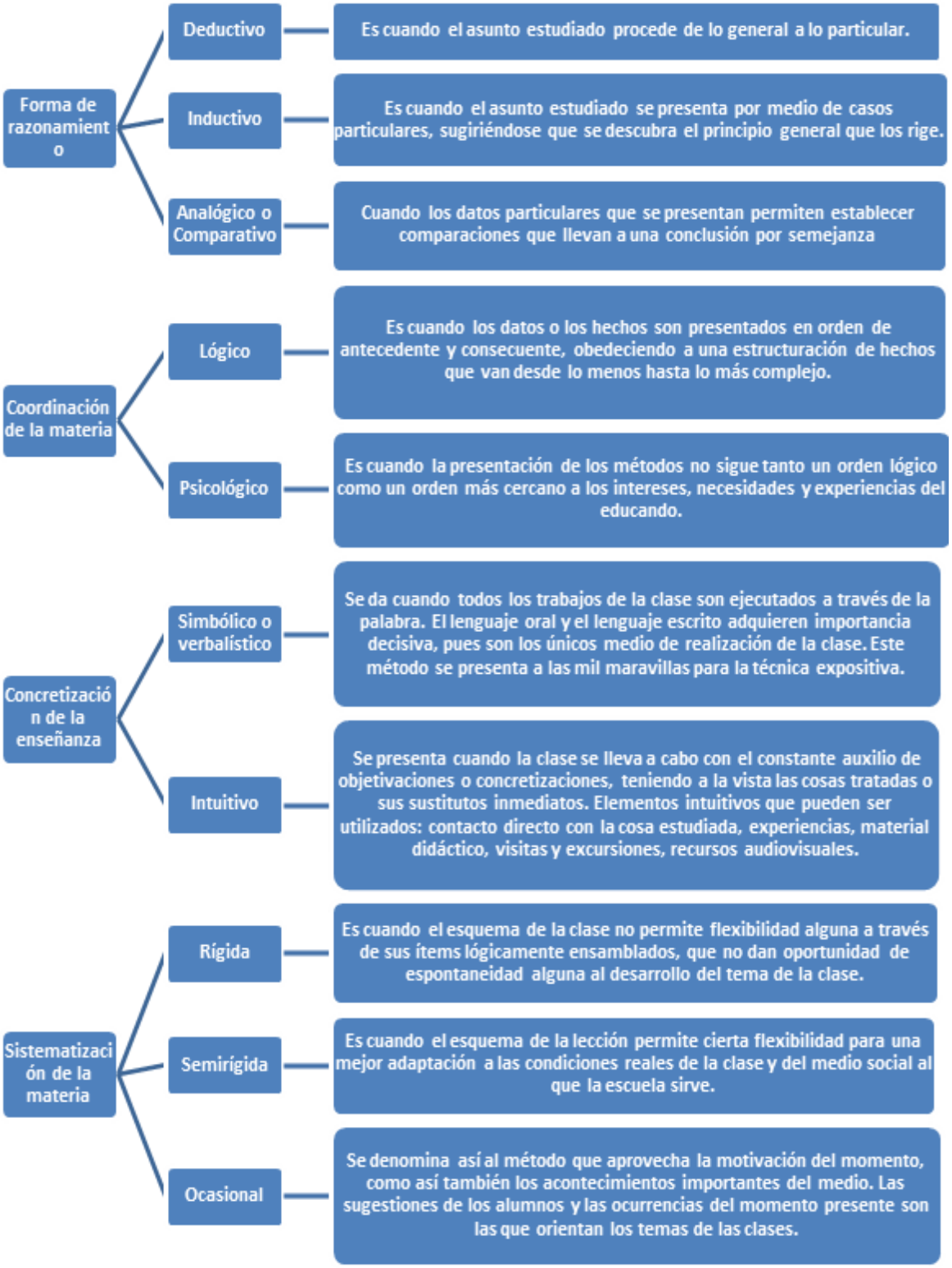


Figura 4. Tipos de métodos (I)

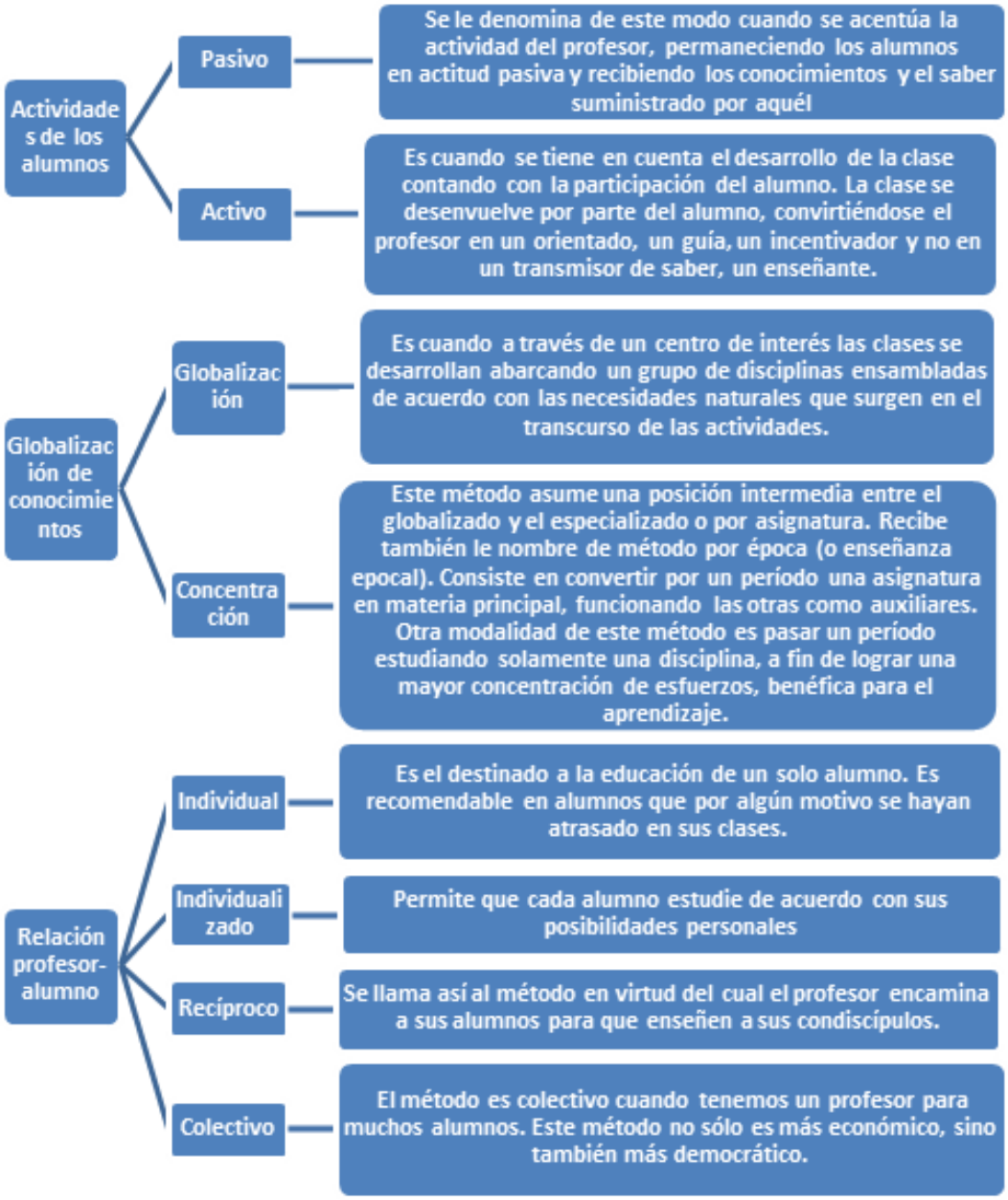


Figura 5. Tipos de Métodos (II)

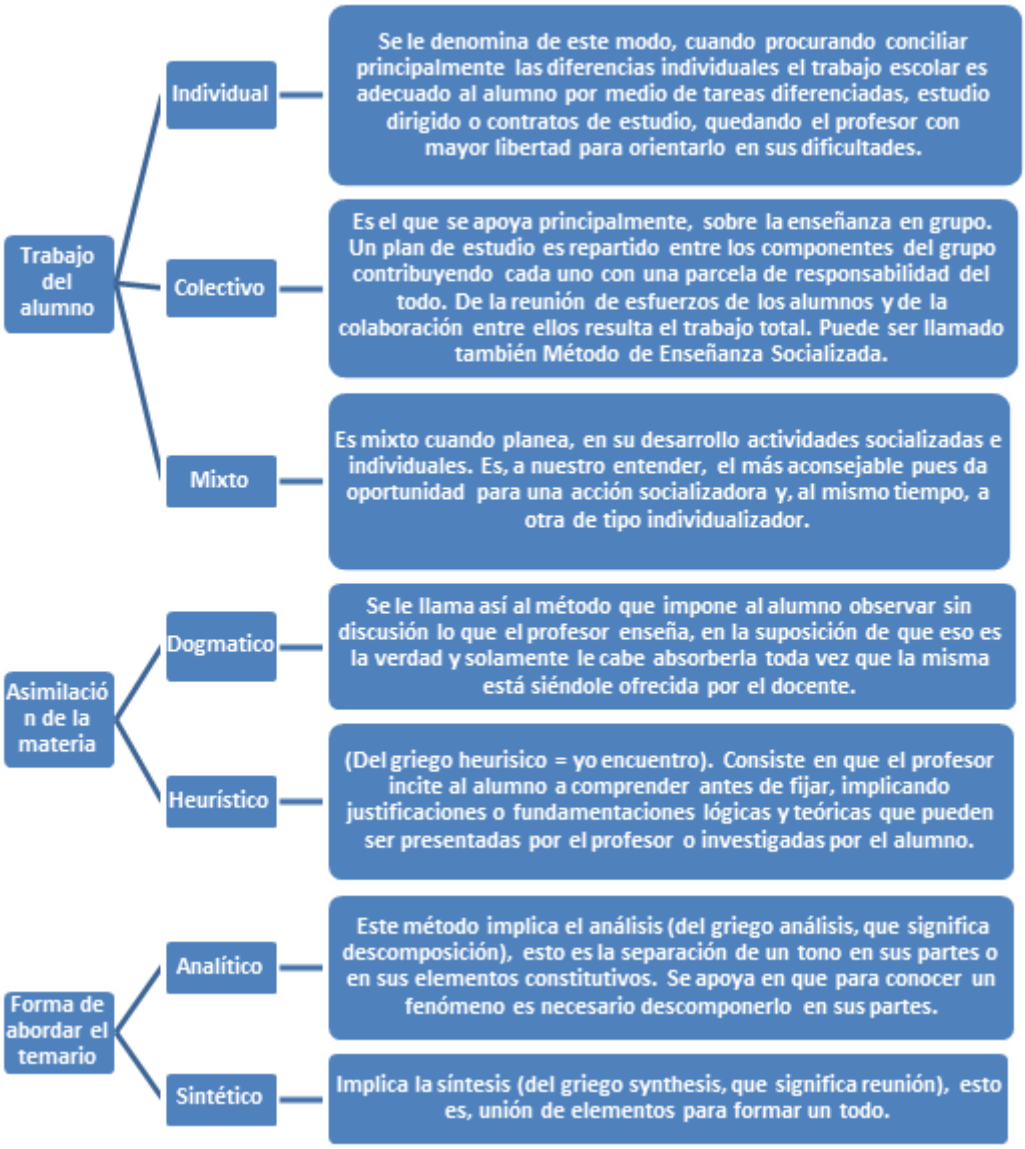


Figura 6. Tipos de Métodos (III)

Cada docente debe hacer el ejercicio práctico de adaptar los principios y métodos de enseñanza citados anteriormente para la creación de un estilo de enseñanza propio, prestando la misma atención a la evaluación que a los problemas y dificultades que presente el alumno [PED].

El docente será el responsable de la formación, tanto individual como grupal de los alumnos, y su labor será facilitar la organización mental de sus alumnos teniendo claros los objetivos individuales de cada uno.

Métodos de Aprendizaje Autodirigido

Según cita Miguel Rebollo en su Propuesta de trabajo para la asignatura "Usuario de WWW" [REB]:

> *"El aprendizaje autodirigido es un método de enseñanza en el que el estudiante asume la iniciativa en el diagnóstico de sus necesidades de aprendizaje, la*

formulación de los objetivos, la elección y búsqueda de los recursos humanos y materiales para el aprendizaje, selecciona las estrategias para mejor aprender y evalúa los resultados obtenidos. El profesor actúa como facilitador y es un recurso más del aprendizaje autodirigido."

El aprendizaje autodirigido, por consiguiente, es un método de enseñanza en el que es el propio estudiante quien toma la iniciativa en el diagnóstico de sus necesidades de aprendizaje, sus objetivos y la selección de los recursos humanos y materiales a emplear para su propio aprendizaje, el alumno es el que establece su estrategia para mejorar y evaluar los resultados obtenidos. El profesor actúa como un apoyo y un recurso más del método.

2.3.3. Soluciones actuales para el aprendizaje de la programación

Analizamos las soluciones existentes para el problema que supone la comprensión de errores de programación por parte del alumno, así como las limitaciones que ofrecen.

Según publicó Juan Ramón Pérez Pérez en su Tesis "CLASIFICACION DE USUARIOS BASADA EN LA DETECCION DE ERRORES USANDO TECNICAS DE PROCESADORES DE LENGUAJE" las soluciones actuales para el problema de la detección y aprendizaje de errores son [PER]:

- **Sistemas de aprendizaje virtual de la programación:**
  - Entornos analizados:
    - *Algorithms in action*, JCAT, HalVis, KBS-*Hyperbook Introduction to Java Programming, Curso interactivo de programación en Pascal, Exploring Computer Science Concepts with Scheme, ELM-ART, WWW-Based C++ Course, DSTool*
  - Ventajas
    - Web como medio de comunicación entre el estudiante y el sistema.
  - Inconvenientes:
    - Son demasiado teóricos.
    - Muy pocos disponen de herramientas de programación.
    - Los ejemplos de código alejados de los problemas reales
- **Entornos para el aprendizaje de la programación**
  - Entornos analizados:
    - BlueJ [Kölling 2003],
    - AnimPascal [Satratzemi 2001]
  - Inconvenientes
    - No proporcionan ayuda:

        - Sobre la interpretación de los errores cometidos,
        - Sobre la solución de los mismos.
    - Necesitan instalación y configuración por parte del usuario.
    - Los estudiantes tienen problemas en la transición hacia entornos comerciales

❑ **Entornos de desarrollo comerciales**

- Entornos analizados:
    - Eclipse, NetBeans, JBuilder
- Inconvenientes
    - Gestión de errores pobre.
        - No señalan errores conceptuales.
        - Es necesaria una experiencia para poder relacionar los síntomas de los defectos con sus verdaderas causas para poder repararlos.
        - Información ambigua.
        - No siempre la información del error es correcta.
    - No proporcionan ayuda para solucionar errores.
    - Complejos para programadores principiantes

❑ **Entornos colaborativos**

- Entornos analizados
    - RECIPE, (*Real-time Collabortive Interactive Programming Environment*) [Shen 2000]
    - COLLEGE [Bravo C. 2004], PlanEdit [Redondo 2002]
- Ventajas
    - Facilita la revisión del código por personas distintas al autor lo que permite detectar y eliminar.
    - Permiten realizar el trabajo de forma síncrona.
- Inconvenientes
    - No utilizan la información de los errores para generar conocimiento.
    - Muchas veces es difícil aprovechar el conocimiento aportado por un usuario para reutilizarlo en un futuro.

❑ **Gestores de prácticas avanzados**

- Sistemas analizados:
    - Sistema de entrega y evaluación de objetivos de aprendizaje [Huizinga 2001], Praktomat [Zeller 2000]
- Inconvenientes
    - Informe sobre errores de compilación y prueba
        - Información proporcionada no es muy reveladora de los verdaderos problemas
    - Protocolo petición-respuesta

- Sistema sólo ayuda a corregir errores; pero no a evitarlos.
- No permite revisar del proceso de desarrollo que ha seguido el alumno.

❑ **Herramientas de detección de errores**

- Inspección manual
  - Técnica muy eficiente.
  - Consumen tiempo y se requiere práctica para aplicarlas correctamente.
- Técnicas dinámicas (pruebas y asertos)
  - Técnica imprescindible para comprobar que se cumplen los requisitos.
  - Es difícil construir un conjunto de prueba adecuado en programas complejos.
- Técnicas estáticas
  - No están limitados a la calidad de los casos de prueba.
  - El análisis no siempre es suficientemente preciso y excesiva cantidad de información poco relevante.

❑ **Herramientas para la detección automática de errores**

- Pueden ahorrar tiempo
- Generan gran cantidad de *warnings* que son difíciles de analizar para un usuario

❑ **Herramientas que buscan errores / herramientas que comprueban el estilo.**

Otras herramientas disponibles para la detección de errores son:

Nombre	Versión	Entrada	Interfaces	Técnica de análisis	Comprobaciones
JLint	3.0	Bytecode	LC	Sintaxis, Flujo de datos	Incorrecciones, problemas de concurrencia
FindBugs	0.7.3 (2004)	Bytecode	LC, GUI, IDE, Ant	Sintaxis (patrones de error), Flujo de datos	Incorrecciones, problemas de concurrencia, mal rendimiento
PMD	1.6	Fuente	LC, GUI, IDE, Ant	Sintaxis (patrones de error)	Incorrecciones, normas de estilo
Checkstyle		Fuente	LC, Ant	Sintaxis	Normas de estilo
JCSC			LC, Ant	Sintaxis	Normas de estilo
DoctorJ			LC	Sintaxis	Coherencia entre documentación y código
Jwiz		Fuente	LC	Sintaxis	Incorrecciones

*Figura 7. Otras herramientas disponibles para detección de errores [**PER**]*

3. ANÁLISIS DE LA HERRAMIENTA

Una vez tratada la parte más teórica del proyecto, en este apartado se formalizan los pasos a seguir en la elaboración de la herramienta teniendo en cuenta el marco regulador en el que se ajusta dicha herramienta, la definición del alcance y los requisitos de usuario y de software que se cumplen.

3.1. Marco Regulador

Para definir el marco regulador correspondiente al proyecto se debe tener en cuenta que la gestión de los errores descritos anteriormente determina la calidad del software, esta labor es práctica habitual en el proceso de desarrollo de cualquier software.

> *"Se conoce como SQA (Software Quality Assurance) o GCS (Gestión de la Calidad del Software) al grado con el que un sistema, componente o proceso cumple las necesidades o expectativas del cliente o usuario."*
>
> IEEE Std. 610-1990 [IEE]

Dentro de este ámbito, la primera norma europea que se utiliza como marco de referencia en cuestiones de calidad y gestión de la formación tanto para los estudiantes como para los diseñadores de proyectos eLearning es la norma CEN/TC 353 "Tecnologías de la información y la comunicación para la educación y el aprendizaje" [AEN]. Esta norma es una adaptación de otra norma internacional de calidad ISO / IEC 19796-1 [ISO] en el ámbito europeo.

La norma CEN/TC 353 puede aplicarse a múltiples escenarios de aprendizaje ya que puede adaptarse a la formación continua, al entorno profesional y al espacio europeo de educación superior y proporciona un marco común para describir, entender y especificar propiedades y métricas críticas de la calidad. El Marco de Referencia para la Descripción de Propuestas de Calidad (MRDPC) es un modelo de proceso elaborado y detallado. Este trabajo de normalización armoniza conceptos, especificaciones, términos y definiciones existentes para aprendizaje, educación y formación [ISI].

Por otra parte, hay que tener en cuenta la legislación vigente en la educación superior universitaria, cita el BOE Ley Orgánica 6/2001, de 21 de diciembre de Universidades en su epígrafe VII:

> *"El auge de la sociedad de la información, el fenómeno de la globalización y los procesos derivados de la investigación científica y el desarrollo tecnológico están transformando los modos de organizar el aprendizaje y de generar y transmitir el conocimiento. En este contexto, la Universidad debe liderar este proceso de cambio y, en consecuencia, reforzar su actividad investigadora para configurar un modelo que tenga como eje el conocimiento. La Ley otorga, mediante un título propio, carta de naturaleza a la actividad investigadora en la Universidad. Lo anteriormente expuesto está en consonancia con el manifiesto compromiso de*

> *los poderes públicos de promover y estimular, en beneficio del interés general, la investigación básica y aplicada en las Universidades como función esencial de las mismas, para que las innovaciones científicas y técnicas se transfieran con la mayor rapidez y eficacia posibles al conjunto de la sociedad y continúen siendo su principal motor de desarrollo […]"*
>
> (BOE 24-12-2001) [BOE]

Con respecto al auto-aprendizaje, según el Real Decreto 1538/2006, Capítulo IV "Definición del currículo por las Administraciones educativas", en su Artículo 18. "Adaptación al entorno socio-productivo", Aptdo. 2:

> *"Las Administraciones educativas, con el fin de facilitar al alumnado la adquisición de las competencias del título correspondiente, en el marco de sus competencias, promoverán la autonomía pedagógica organizativa y de gestión de los centros que impartan formación profesional, fomentarán el trabajo en equipo del profesorado y el desarrollo de planes de formación, investigación e innovación en su ámbito docente y las actuaciones que favorezcan la mejora continua de los procesos formativos."*
>
> (BOE-A-2007-92)[BO2]

3.2. Definición del Sistema

A continuación se define el funcionamiento de la herramienta diseñada partiendo de los objetivos establecidos tanto para el alumno como para el profesor.

3.2.1. Alcance

La herramienta está diseñada para multiusuario, es decir, proporciona elementos de auto-evaluación al alumno y mejora el proceso de evaluación del profesor fomentando y mejorando el aprendizaje mediante detección de los errores de programación en un entorno multilenguaje.

3.2.2. Entorno de Desarrollo

El diseño del proyecto se ha desarrollado enteramente en lenguaje Java, en el entorno de desarrollo Eclipse.

Para el entorno gráfico, se ha importado la librería necesaria para instalación de WindowBuilder que nos permite desarrollar una Interfaz Gráfica de Usuario (GUI) que permite crear JFrame.

Para la comunicación con la base de datos en Excel se han importado las librerías POI de Apache, cuyo objetivo es crear y mantener las API de Java para manipular varios formatos de archivo basados en los estándares XML abiertos de Office (OOXML) y

formato de Documento Compuesto de Microsoft OLE 2 (OLE2). En resumen, se puede leer y escribir archivos de MS Excel utilizando Java. Además, puede leer y escribir archivos de MS Word y MS PowerPoint utilizando Java. Apache POI es su solución Java Excel (para Excel 97-2008).

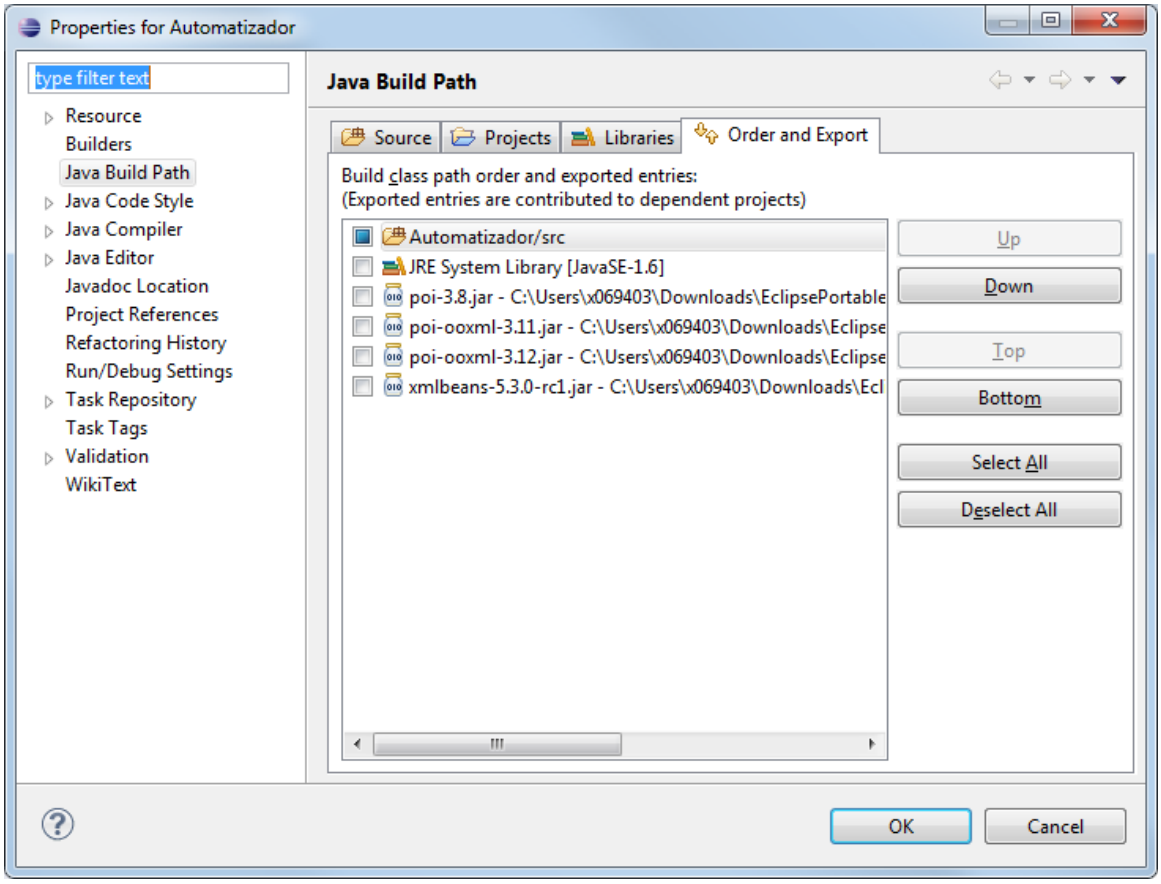


Figura 8. Librerías importadas en Eclipse para la implementación de la herramienta

3.2.3. Requisitos de la Herramienta

Recogen las condiciones mínimas que debe cumplir la herramienta dependiendo del perfil de usuario que actúe sobre dicha herramienta.

Cada requisito se representa con la siguiente tabla tipo:

Identificador: RW-XX			
Título			
Descripción			
Prioridad	Baja ☐	Media ☐	Alta ☐
Fuente	Profesor ☐	Alumno ☐	
Necesidad	Obligatorio ☐	Deseable ☐	Opcional ☐
Estabilidad	Modificable ☐	Inalterable ☐	

Tabla 1. Plantilla de Requisitos de Usuario-Software

- **Identificador** tiene el siguiente formato:
  - R: Requisito
  - W: En caso de ser un Requisito Funcional se completará con “F”, en otro caso “NF”
  - XX: representa la numeración asignada de modo secuencial.

- **Título** indica el nombre del requisito.
- **Descripción** contiene un breve resumen del objetivo del requisito.
- **Prioridad** indica la criticidad del requisito.
- **Necesidad** indica el grado de obligatoriedad del requisito.
- **Estabilidad** indica la posibilidad de modificar el requisito.

Requisitos Funcionales

Identificador: RF-00						
Título	**Pantalla de Configuración**					
Descripción	La herramienta permite al usuario iniciar la sesión accediendo a la interfaz de usuario					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 2. RF-00: Pantalla de Configuración

Identificador: RF-01						
Título	**Login de usuario**					
Descripción	El usuario identificado mediante password en la herramienta accederá a los campos de asignación de pesos según el tipo de error para la evaluación del código					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 3.RF-01: Login de Usuario

Identificador: RF-02						
Título	**Selección de Lenguaje de Programación**					
Descripción	Permite introducir el lenguaje de programación del código a revisar					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 4. RF-02: Selección de Lenguaje de Programación

Identificador: RF-03						
Título	**Resultados de la evaluación**					
Descripción	La ejecución de la herramienta genera un documento donde se especifican los errores que aparecen en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 5. RF-03: Resultados de la evaluación

Identificador: RF-04						
Título	**Documentación automática**					
Descripción	La ejecución de la herramienta genera un documento donde se guardan los comentarios de los programas a evaluar					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 6. RF-04: Documentación automática

Identificador: RF-05						
Título	**Legibilidad - Indentación**					
Descripción	El código debe estar correctamente indentado, según la profundidad de cada instrucción dentro del conjunto de bloques, ya sea mediante espacios en blanco o tabuladores. El método elegido deberá ser aplicado uniformemente a lo largo de todo el programa					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 7. RF-05: Legibilidad-Indentación

Identificador: RF-06						
Título	**Legibilidad - Estilo**					
Descripción	Los nombres de variables, constantes, procedimientos y funciones deberán seguir un formato determinado a lo largo de todo el programa					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 8. RF-06: Legibilidad-Estilo

Identificador: RF-07						
Título	**Legibilidad - Comentarios**					
Descripción	El código debe estar comentado. Se deben incluir comentarios al principio del programa y antes de cada subrutina.					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 9. RF-07: Legibilidad-Comentarios

Identificador: RF-08						
Título	**Legibilidad - Salto incondicional**					
Descripción	No se permite el uso de sentencias de salto incondicional.					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 10. RF-08: Legibilidad-Salto incondicional

Identificador: RF-09						
Título	**Legibilidad - Declaración implícita**					
Descripción	No se permite el uso de declaración implícita de variables					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 11. RF-09: Legibilidad- Declaración implícita

Identificador: RF-10						
Título	**Legibilidad - Formato**					
Descripción	El formato de la definición de variables, constantes, procedimientos y funciones deberán seguir un formato determinado a lo largo de todo el programa.					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 12. RF-10:Legibilidad-Formato

Identificador: RF-11						
Título	**Modularidad - Tamaño de módulo**					
Descripción	El programa debe estar lo suficientemente modulado. La forma de controlar esto será restringiendo el tamaño de los subprogramas					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 13. RF-11: Modularidad-Tamaño de módulo

Identificador: RF-12						
Título	**Modularidad - Valor retorno subprogramas**					
Descripción	Hay que distinguir entre funciones y procedimientos. Aquellos subprogramas que devuelvan un valor serán funciones y el resto serán procedimientos					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 14. RF-12: Modularidad-Valor retorno subprogramas

Identificador: RF-13						
Título	**Modularidad - Uso de argumentos por valor o referencia**					
Descripción	Los argumentos de una subrutina pueden ser pasados por valor o referencia, dependiendo de si se va a querer modificar su contenido dentro de la subrutina o no					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 15. RF-13: Modularidad-Uso de argumentos por valor o referencia

Identificador: RF-14						
Título	**Modularidad - Declaración de argumentos**					
Descripción	No se debe modificar una variable pasada por valor. Los argumentos pasados por referencia deben ser modificados.					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 16.RF-14: Modularidad-Declaración de argumentos

Identificador: RF-15						
Título	**Modularidad - Interfaz**					
Descripción	Relacionados con la interfaz de usuario, es necesario controlar dos aspectos básicos. o No se debe mezclar la lógica del programa con la interfaz de usuario. Esto es, un subprograma que calcule o modifique un dato nunca deberá imprimir por pantalla, y viceversa. o Siempre que se pida un dato por teclado, deberá haber un mensaje previo de petición del dato.					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 17.RF.15: Modularidad-Interfaz

Identificador: RF-16						
Título	**Uso de variables - No inicializada**					
Descripción	Se debería por tanto avisar en caso de encontrar acceso al valor de una variable no inicializada					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 18. RF.16: Uso de variables-No inicializada

Identificador: RF-17						
Título	**Uso de variables - No usada**					
Descripción	Se debería por tanto avisar en caso de encontrar declaración de una variable que no se utiliza nunca					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 19.RF-17: Uso de variables-No usada

Identificador: RF-18						
Título	**Uso de variables - Asignación doble**					
Descripción	Se debería por tanto avisar en caso de asignación sobre una variable asignada anteriormente, sin acceder a su valor entre ambas asignaciones					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 20. RF-18: Uso de variables-Asignación doble

Identificador: RF-19						
Título	**Uso de variables - Acceso a variable global**					
Descripción	Se debería por tanto avisar en caso de acceso a variables globales por visibilidad					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 21. RF-19: Uso de variables-Acceso a variable global

Identificador: RF-20						
Título	**Uso de variables - Doble referencia**					
Descripción	Impedir el paso de argumentos con doble referencia					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☐	Inalterable	☑		

Tabla 22. RF-20: Uso de variables-Doble referencia

Requisitos No Funcionales

Identificador: RNF-00						
Título	**Ruta de Base de Datos**					
Descripción	Campo para introducir la ruta donde se encuentra la Excel de la base de datos de la herramienta					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 23. RNF-00: Ruta de Base de Datos

Identificador: RNF-01						
Título	**Ruta de Ficheros Programas**					
Descripción	Campo para introducir la ruta donde se encuentran los ficheros a evaluar					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 24.RNF-01: Ruta de Ficheros Programas

Identificador: RNF-02						
Título	**Ruta de Resultados**					
Descripción	Campo para introducir la ruta donde se encuentran los resultados de la ejecución de la herramienta					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☑		
Necesidad	Obligatorio	☑	Deseable	☐	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 25.RNF-02: Ruta de Resultados

Identificador: RNF-03						
Título	**Campo Indentación**					
Descripción	Campo para introducir el peso que tiene un error de indentación en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 26.RNF-03: Campo indentación

Identificador: RNF-04						
Título	**Campo Comentario**					
Descripción	Campo para introducir el peso que tiene un error de Comentario en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 27.RNF-04: Campo Comentario

Identificador: RNF-05						
Título	**Campo Estilo**					
Descripción	Campo para introducir el peso que tiene un error de Estilo en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 28.RNF-05: Campo Estilo

Identificador: RNF-06						
Título	**Campo Salto Incondicional**					
Descripción	Campo para introducir el peso que tiene un error de Salto incondicional en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 29.RNF-06: Campo Salto incondicional

Identificador: RNF-07						
Título	**Campo Declaración Implícita de variables**					
Descripción	Campo para introducir el peso que tiene un error de Declaración implicita de variables en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 30.RNF-07: Campo Declaración Implícita de variables

Identificador: RNF-08						
Título	**Campo Formato de variables y constantes**					
Descripción	Campo para introducir el peso que tiene un error de formato de variables y constantes en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 31.RNF-08: Campo Formato de variables y constantes

Identificador: RNF-09						
Título	**Campo Tamaño máximo de un módulo**					
Descripción	Campo para introducir el tamaño máximo que debe tener un módulo (subprograma o método)					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 32.RNF-09: Campo Tamaño máximo de un módulo

Identificador: RNF-10						
Título	**Campo Tamaño de módulo**					
Descripción	Campo para introducir el peso que tiene un error de sobrepaso de tamaño de módulo en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 33.RNF-10: Campo Tamaño de módulo

Identificador: RNF-11						
Título	**Campo Valor retorno subprograma**					
Descripción	Campo para introducir el peso que tiene un error de valor de retorno de subprograma de tamaño de módulo en la					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 34.RNF-11: Campo Valor retorno subprograma

Identificador: RNF-12						
Título	**Campo Uso de argumentos (Valor / Referencia)**					
Descripción	Campo para introducir el peso que tiene un error de uso de argumentos (valor o referencia) cuando el subprograma no acepta argumentos de entrada o salida en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 35.RNF-12: Campo Uso de argumentos (Valor/Referencia)

Identificador: RNF-13						
Título	**Campo Declaración de Argumentos**					
Descripción	Campo para introducir el peso que tiene un error en la declaración de argumentos en la programación					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 36.RNF-13: Campo Declaración de argumentos

Identificador: RNF-14						
Título	**Campo Interfaz**					
Descripción	Campo para introducir el peso que tiene un error por mezclar la interfaz con la lógica del programa					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 37.RNF-14: Campo Interfaz

Identificador: RNF-15						
Título	**Campo No inicializada**					
Descripción	Campo para introducir el peso que tiene un error de uso de variable no inicializada					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 38.RNF-15: Campo No inicializada

Identificador: RNF-16						
Título	**Campo No usada**					
Descripción	Campo para introducir el peso que tiene un error por no usar una variable declarada					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 39.RNF-16: Campo No usada

Identificador: RNF-17						
Título	**Campo Asignación doble**					
Descripción	Campo para introducir el peso que tiene un error por no asignación doble de variables					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 40.RNF-17: Campo Asignación doble

Identificador: RNF-18						
Título	**Campo Acceso a var. Global**					
Descripción	Campo para introducir el peso que tiene un error de acceso a una variable global					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 41. .RNF-18: Campo Acceso a var. Global

Identificador: RNF-19						
Título	**Campo Doble referencia**					
Descripción	Campo para introducir el peso que tiene un error de doble referencia					
Prioridad	Baja	☐	Media	☐	Alta	☑
Fuente	Profesor	☑	Alumno	☐		
Necesidad	Obligatorio	☐	Deseable	☑	Opcional	☐
Estabilidad	Modificable	☑	Inalterable	☐		

Tabla 42.RNF-19: Campo Doble referencia

3.2.4. Casos de Uso

Un caso de uso es una unidad funcional coherente de un sistema, aplicación o clase, en donde uno o más actores interaccionan con el sistema que ejecuta las acciones [VEG].

Consta de los siguientes elementos:

- **Actores**: cualquier elemento externo al sistema que interacciona con él. Un actor puede interpretar distintos roles dependiendo del caso de uso en que participe. Un actor inicia el caso de uso para solicitar, modificar o informar al sistema de un suceso que le incumbe.

- **Casos de uso**: son los principales procesos end to end realizados por los actores en su relación con el sistema. Cada caso es distinto y completo en sí mismo dentro de la utilización de una aplicación o sistema. En él se describen

los pasos a seguir en un uso del sistema.

- **Relaciones**: muestra la comunicación entre actor-sistema y sistema-actor.

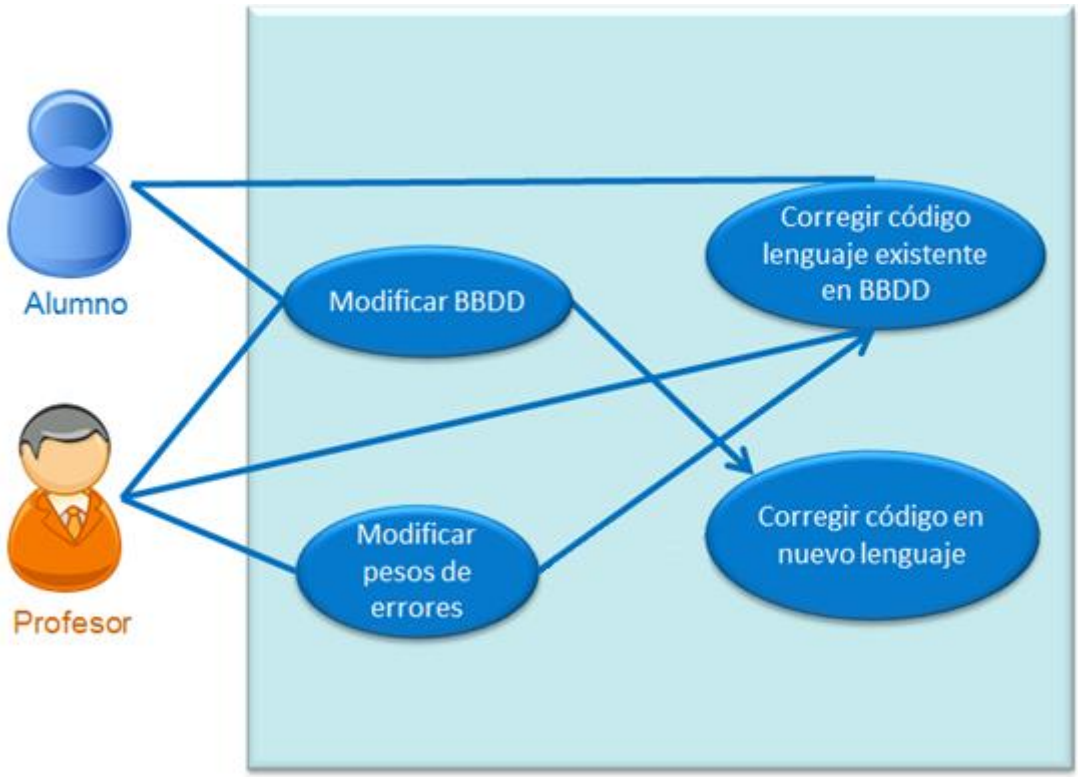


Figura 9. Casos de uso de la herramienta

Cada caso de uso se representa con la siguiente tabla tipo:

Identificador: CU-XX	
Título	
Descripción	
Actores	Profesor ☐ Alumno ☐
Precondiciones	
Secuencia normal	
Secuencia alternativa	
Importancia	Baja ☐ Media ☐ Alta ☐
Frecuencia esperada	

Tabla 43. Plantilla de casos de uso

- **Identificador** tiene el siguiente formato:
  - CU: Caso de Uso
  - XX: representa la numeración asignada de modo secuencial.
- **Título** indica el nombre del caso de uso.
- **Descripción** contiene un breve resumen del objetivo del caso de uso.
- **Actores:** indica el rol del usuario involucrado en el caso de uso

(profesor/alumno)

- **Precondiciones**: Requisitos previos que deben cumplirse antes de comenzar el caso de uso actual
- **Secuencia normal:** Secuencia normal, o caso positivo. Se indicaran los pasos a seguir secuencialmente.
- **Secuencia alternativa**: Secuencia inusual o inesperada, o caso negativo. Se indicarán los pasos a seguir secuencialmente.
- **Importancia:** indica la criticidad del requisito.
- **Frecuencia esperada:** indica el grado de ocurrencia del caso de uso en la herramienta.

Identificador: CU-01	
Título	**Corregir código en lenguaje existente en la base de datos**
Descripción	El usuario introduce lenguaje existente en la base de datos
Actores	Profesor ☑ Alumno ☑
Precondiciones	El usuario introdujo el resto de campos obligatorios la ruta de la BBDD y las rutas de ficheros de origen y resultados.
Secuencia normal	1. El usuario pulsa el botón Aceptar 2. La herramienta muestra el resultado de la corrección informando del número de errores y la puntuación obtenida por módulo
Secuencia alternativa	1. El usuario introduce un password incorrecto 2. La herramienta no habilita los campos de la seccion de puntuación de uso restringido
	1. El usuario introduce alguna ruta incorrecta en los campos obligatorios 2. La herramienta avisa al usuario de que el fichero no existe
Importancia	Baja ☐ Media ☐ Alta ☑
Frecuencia esperada	Es el caso ideal, se espera que un usuario habitual de la herramienta use siempre esta caso

Tabla 44. CU-01. Corregir código en lenguaje existente en la base de datos

Identificador: CU-02	
Título	**Modificar la Base de datos**
Descripción	El usuario introduce accede a la Excel de datos e introduce los campos indicados para el nuevo lenguaje
Actores	Profesor ☑ Alumno ☑
Precondiciones	El usuario introdujo en el campo lenguaje el nombre del lenguaje de programación y un aviso le indicó que debía introducir los valores del nuevo lenguaje en la BBDD.
Secuencia normal	1. El usuario introduce los valores en una nueva hoja en la BBDD 2. El usuario pulsa en el botón Aceptar 3. La herramienta muestra el resultado de la corrección informando del número de errores y la puntuación obtenida
Secuencia alternativa	1. El usuario introduce alguna ruta incorrecta en los campos obligatorios 2. La herramienta avisa al usuario de que el fichero no existe
	1. El usuario introduce el nuevo lenguaje incorrectamente 2. La herramienta avisa al usuario de que el lenguaje no existe 3. El usuario debe verificar que una hoja excel de la ruta especifiacada de BBDD tiene el mismo nombre que el introducido en la herramienta
Importancia	Baja ☐ Media ☑ Alta ☐
Frecuencia esperada	Este caso sucede siempre que se desea corregir un nuevo lenguaje. Un usuario experimentado de la herramienta puede saltarse este caso e introducir el lenguaje sin esperar a que salte el aviso

Tabla 45. CU-02. Modificar la Base de datos

Identificador: CU-03	
Título	**Corregir código en nuevo lenguaje**
Descripción	El usuario introduce lenguaje existente en la base de datos
Actores	Profesor ☑ Alumno ☑
Precondiciones	1. El usuario introdujo el resto de campos obligatorios la ruta de la BBDD y las rutas de ficheros de origen y resultados. 2. El usuario modificó la base de datos
Secuencia normal	1. El usuario pulsa el botón Aceptar 2. La herramienta muestra el resultado de la corrección informando del número de errores y la puntuación obtenida por módulo
Secuencia alternativa	1. El usuario introduce alguna ruta incorrecta en los campos obligatorios 2. La herramienta avisa al usuario de que el fichero no existe
Importancia	Baja ☐ Media ☐ Alta ☑
Frecuencia esperada	Este caso también lo utilizarán los usuarios que modifiquen la base de datos, aunque el lenguaje ya exista. De modo que este caso será utilizado para usuarios conocedores de las posibilidades de mantenibilidad de la herramienta. Presumiblemente profesores que añadan temario en su asignatura y deseen añadir más sentencias al lenguaje de estudio

Tabla 46. CU-03. Corregir código en nuevo lenguaje

Identificador: CU-04	
Título	**Modificar pesos de errores**
Descripción	El usuario modifica los valores en la sección Puntuación por tipos de error en la interfaz
Actores	Profesor ☑ Alumno ☐
Precondiciones	El usuario introdujo correctamente el password
Secuencia normal	1. El usuario pulsa el botón Modifcar 2. La herramienta habilita los campos de la sección Puntuación por tipos de error en la interfaz 3. El usuario modifica los valores de los pesos de error según la criticidad que considere 4. El usuario pulsa en botón Aceptar 5. La herramienta muestra el resultado de la corrección informando del número de errores y la puntuación obtenida por módulo
Secuencia alternativa	1. El usuario deja vacío alguno de los campos de los pesos 2. La herramienta avisa al usuario de que todos los campos deben ser completados
	1. El usuario introduce alguna ruta incorrecta en los campos obligatorios 2. La herramienta avisa al usuario de que el fichero no existe
Importancia	Baja ☐ Media ☑ Alta ☐
Frecuencia esperada	Este caso se utilizará solo por el usuario administrador de la herramienta, por lo que su uso es bajo

Tabla 47. CU-04. Modificar pesos de errores

4. DISEÑO E IMPLEMENTACIÓN

En este capítulo se detalla el diseño de la herramienta, la arquitectura empleada y los detalles de implementación justificadamente. También se enumerarán las decisiones de diseño tomadas durante la fase de programación del proyecto.

4.1. Arquitectura del sistema

La arquitectura empleada ha sido el Modelo Vista Controlador (MVC) ya que se ajusta a las características de configuración de la aplicación. MVC sigue un patrón de arquitectura en el que se separa la interfaz de usuario con los datos de la aplicación y la lógica de negocio, ver Figura 10 [FRE].

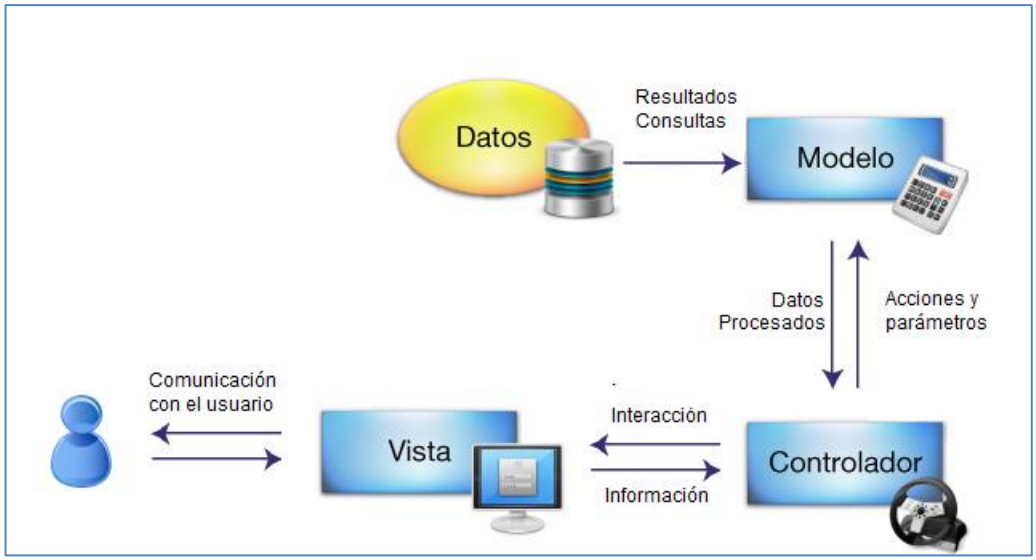


Figura 10. Arquitectura Modelo-Vista-Controlador **[FRE]**

- **El Modelo**: esta capa contiene la lógica de negocio, es decir, el modelo conceptual del sistema. Desde este módulo se accede directamente a la base de datos, actúa como intermediario entre el controlador y la base de datos. En la capa Modelo se incluirán las clases y subsistemas relacionados con los datos que gestiona la aplicación. La capa Modelo no contiene ninguna referencia hacia los otros dos módulos Vista y Controlador. A pesar de ellos, el propio sistema se encarga de relacionar las capas Modelo y Vista. Un cambio en el Modelo se ha de notificar a la Vista. Las peticiones de acceso o manipulación de información llegan al Modelo a través del Controlador.

- **El Controlador**: capa que responde a eventos producto de la interacción del usuario con el sistema a través de la interfaz, recibidos de la Vista, que invoca las peticiones pertinentes a la capa Modelo cuando se hace alguna solicitud sobre la información y datos. De modo que la capa del Controlador hace de intermediario entre las otras dos: Vista y Modelo.

- **La Vista**: capa con la que interactúa el usuario. Presenta el Modelo (lógica de negocio) en un interfaz u otro formato adecuado para interactuar. Esta capa está, por lo general, compuesta por interfaces gráficas que el usuario utilizará para manejar los datos del sistema. La Vista se manda la petición del usuario a la capa Controlador.

Este proyecto se ha diseñado de manera que se ajusta perfectamente a la arquitectura MVC donde Vista es la interfaz con el usuario, con la ventana de configuración de la herramienta que el usuario utiliza. A través del Modelo se realiza la lógica de negocio de la herramienta, se realiza la gestión de la base de datos y la lógica en Java. El Controlador es el responsable de gestionar los eventos de entrada del usuario desde la Vista devolviendo eventos y resultados de la propia herramienta.

Con esta arquitectura se consigue también cumplir con los requisitos solicitados de reutilización de componentes y mantenibilidad de los sistemas para un posterior reuso de la herramienta, además facilidad para desarrollos escalables

De acuerdo a la arquitectura descrita en el apartado anterior, se va a detallar el diseño de la herramienta de aprendizaje dirigido según los bloques indicados, ver Figura 11:

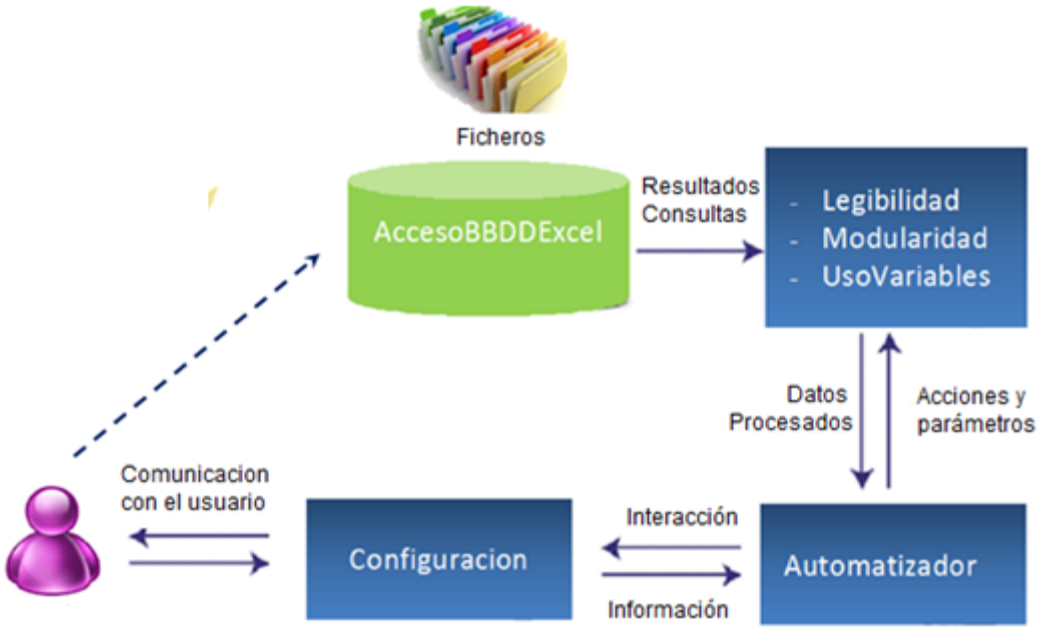


Figura 11. Arquitectura de la Herramienta de Aprendizaje Dirigido

4.2. Diseño de la Interfaz Configuración de Usuario (Vista)

La interfaz de usuario está diseñada para cualquiera de los roles de usuario susceptibles de usar la herramienta, es decir, tanto para alumnos como para profesores.

Como puede verse en la Figura 12, la interfaz consta de una serie de secciones que se

habilitarán o no, según el tipo de usuario.

La ventana de la interfaz de usuario se ha diseñado con un *look and feel* atractivo para el usuario y es única para cualquier rol, se divide en 4 secciones:

1. **Base de datos**, se deberá introducir el lenguaje de programación que deseamos evaluar y la ruta donde tendremos nuestra Excel de datos.
   - Campo **Lenguaje de programación*** (obligatorio, habilitado y editable).
   - Campo **Ruta Base Datos*** (obligatorio, habilitado y editable).

2. **Rutas de ficheros**, donde se deberán introducir las rutas locales de usuario donde se encuentras los ficheros a evaluar, así como la ruta donde se desea guardar los resultados de dicha evaluación:
   - Campo **Ruta origen*** donde se deberá introducir la ruta de los ficheros de entrada que contienen los programas a evaluar (obligatorio, habilitado y editable).
   - Campo **Ruta resultados*** donde se introducirá la ruta donde se desea guardar los resultados de la evaluación, así como la documentación automática de los programas (obligatorio, habilitado y editable).

3. **Login de usuario**: Permite activar los campos de la sección de "Puntuación por tipos de error"
   - Campo **Password*** (obligatorio, habilitado y editable de tipo contraseña)

4. **Puntuación por tipo de error**: sección editable sólo en caso de identificarse mediante la contraseña correcta en el campo Password (inicialmente accesible sólo para el rol profesor):
   - Módulo Legibilidad:
     - Campo **Indentación,** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
     - Campo **Comentario** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
     - Campo **Estilo** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
     - Campo **Salto Incondicional** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
     - Campo **Declaración implícita** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").

- Campo **Formato** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").

- Módulo Modularidad:
  - Campo **Tamaño máx. modulo,** contiene valor máximo que debe tener un subprograma y es asignado por el (inhabilitado para rol Alumno, valor por defecto "100").
  - Campo **Tamaño de módulo** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
  - Campo **Valor retorno subprograma** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
  - Campo **Uso argumentos** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
  - Campo **Interfaz** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
- Módulo Uso de Variables:
  - Campo **No inicializada,** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
  - Campo **No usada** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
  - Campo **Asignación doble** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
  - Campo **Acceso a var. Global** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").
  - Campo **Doble referencia** contiene el peso asignado por el profesor a cada error de este tipo (inhabilitado para rol Alumno, valor por defecto "1").

Por último aparece el botón **Aceptar** con el que comienza la corrección del código indicado por el usuario.

Como ya hemos mencionado en el capítulo 3, para el entorno gráfico, se ha importado la librería necesaria para instalación de *WindowBuilder* en nuestro IDE para visualizar la

ventana de tipo *JFrame* que tiene el siguiente formato:

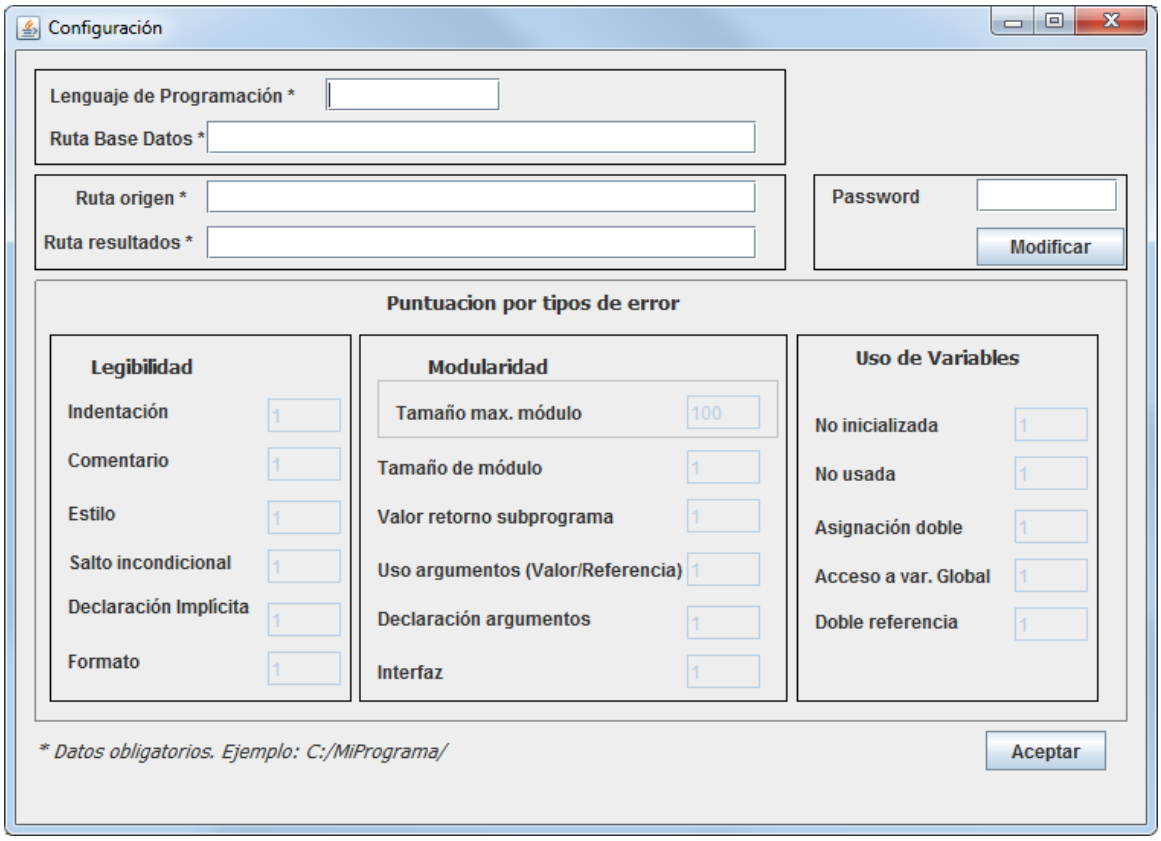


Figura 12. Interfaz de Usuario de la Herramienta

Además se ha añadido la posibilidad de recuperar la sesión anterior, que se ha guardado en un archivo temporal del disco local del usuario, ver Figura 13.

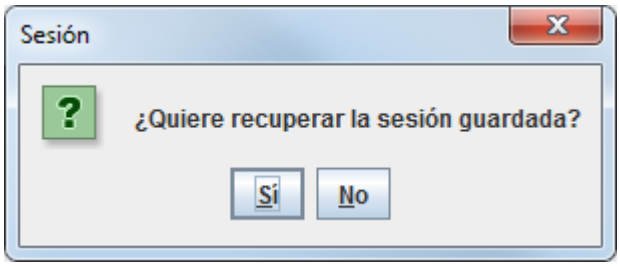


Figura 13. Ventana que ofrece la opción de recuperar sesión

4.2.1. Roles de usuario

Dependiendo del usuario se ofrecen o no determinadas opciones. En el diagrama de secuencia se muestran las distintas opciones de la herramienta, ver Figura 14.

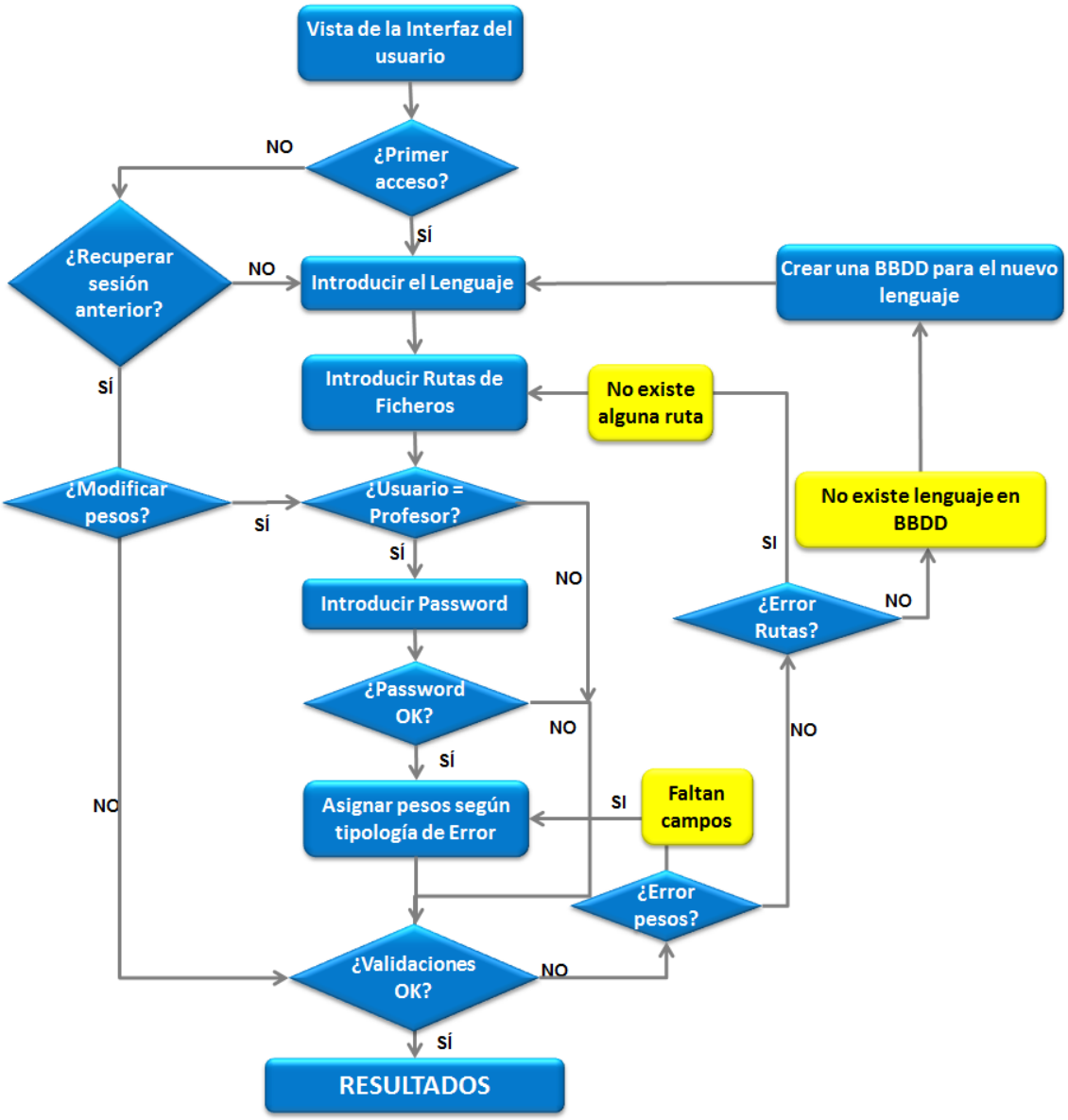


Figura 14. Diagrama de flujo de la herramienta

Rol Alumno: En caso de tratarse de un alumno, lo que se persigue es la autocorrección del código, por lo que no se permite asignación de pesos a cada error. Ver Figuras 15 y 16.

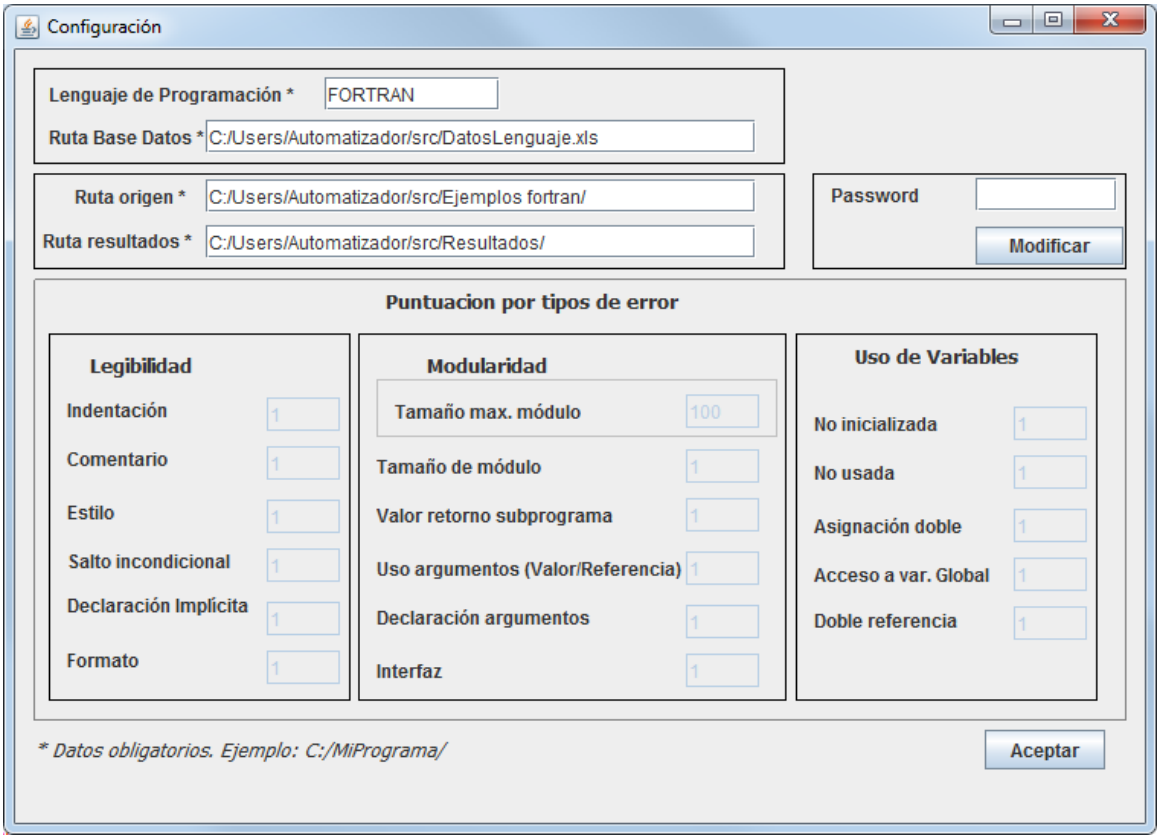


Figura 15. Interfaz de alumno. Sin contraseña

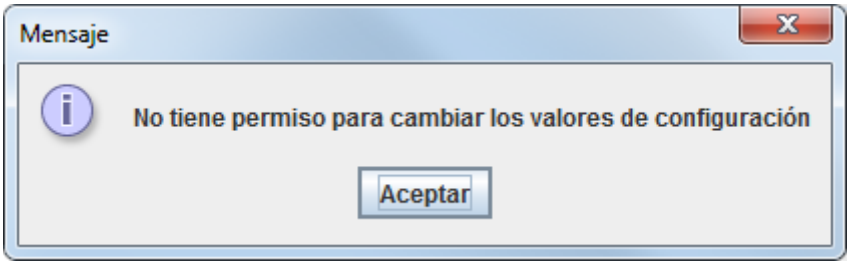


Figura 16. Mensaje de error por error de contraseña

Rol Profesor: En caso de tratarse de un profesor, lo que se persigue es la evaluación del código, por lo que se permite asignación de pesos a cada error, el profesor valorará según las necesidades y los conocimientos del alumno, qué pesos son más o menos críticos dependiendo del avance del curso. Ver Figura 17.

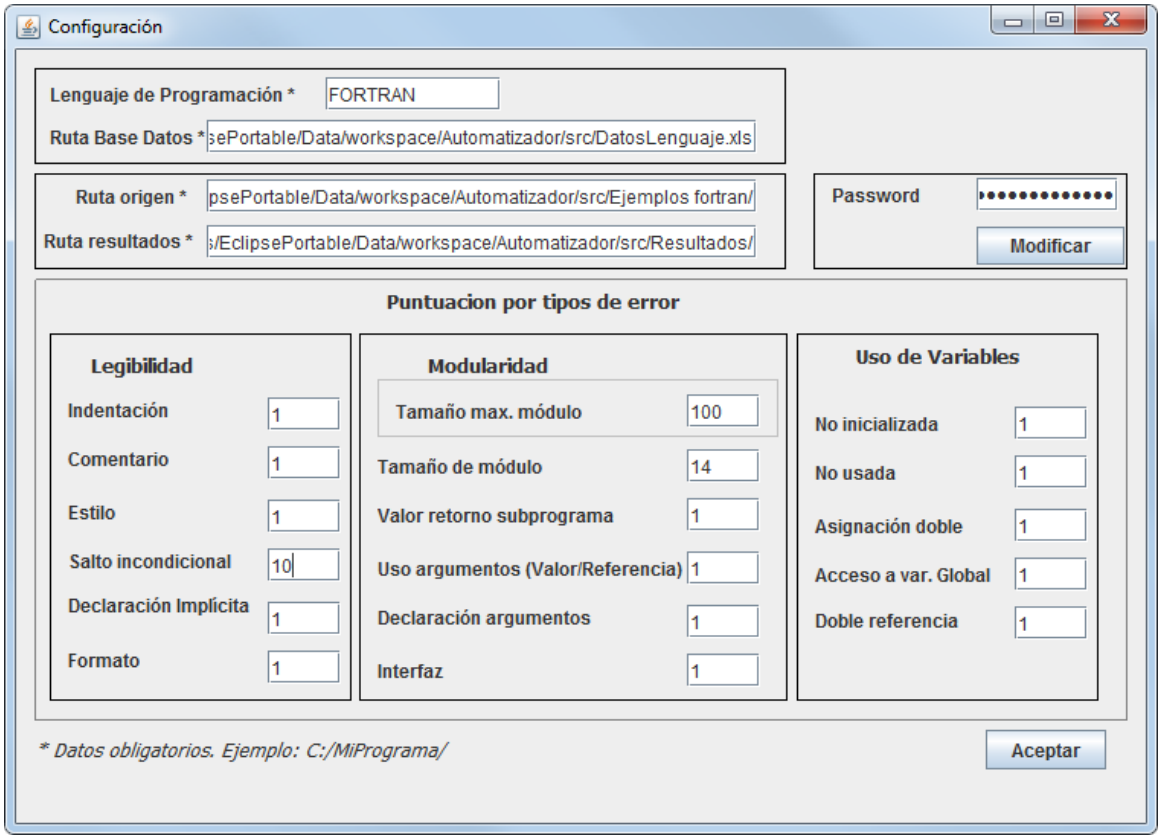


Figura 17. Interfaz de profesor. Con contraseña

4.2.2. Interacción con el Automatizador (Controlador)

Al pulsar el botón Aceptar la capa Visión conecta con la capa Controlador, diseñada en la clase **Automatizacion**, y devuelve como resultado una ventana, ver Figura 18, por cada fichero que contenga la Ruta Origen. Esta ventana contiene el número de errores del fichero indicado y la puntuación de cada módulo.

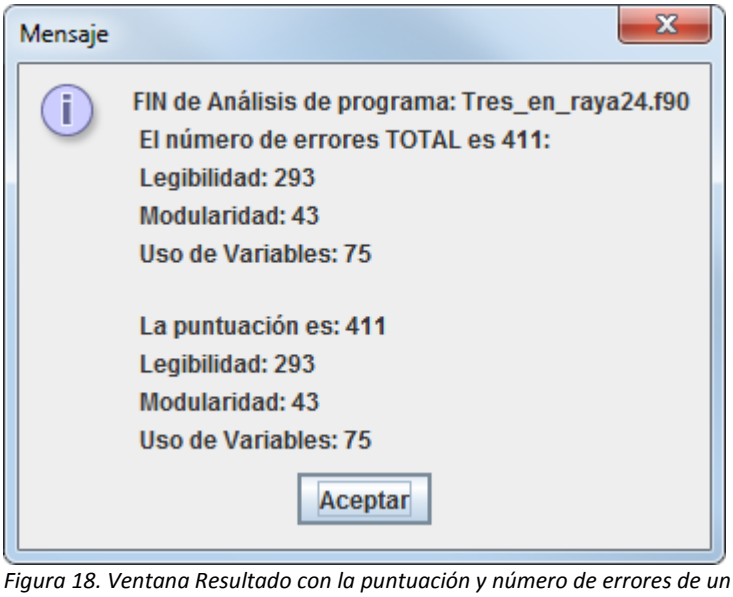


Figura 18. Ventana Resultado con la puntuación y número de errores de un fichero

4.3. Diseño del Automatizador (Controlador)

La capa Controlador está formada por la clase **Automatizacion**, que es la que contiene el método *main* del sistema, es la clase que hace las llamadas al resto de clases contenidas en la capa Modelo, con la lógica de la herramienta.

En el módulo Controlador, por tanto, se hacen llamadas al resto de módulos, es decir, se crean instancias del resto de clases, ver Figura 19.

Automatizacion.java | Legibilidad.java | Modularidad.java | UsoVariables.java | Configuracion.java | AccesoBBDD.java

```
42  btnAceptar.addActionListener(new ActionListener() {
43      public void actionPerformed(ActionEvent e) {
44          //se valida que los campos de la sección de puntuación están completados correctamente
45          if (formulario.txtLenguaje.getText().isEmpty()||formulario.txtRutaBBDD.getText().isEmpty()||formulario.txtRutaOrigen.getText().isEm
46                  ||formulario.textField_mod_tam_max.getText().isEmpty()|| formulario.textField_mod_tamaño.getText().isEmpty()|| formulario.t
47                  || formulario.textField_uv_noinic.getText().isEmpty()|| formulario.textField_uv_no_usada.getText().isEmpty()|| formulario.t
48              JOptionPane.showMessageDialog(null,"Debe completar todos los campos");
49
50
51          }//se valida que los campos en el área de puntuación por tipos de error están en formato número
52          else if (!formulario.textField_leg_indent.getText().matches("[0-9]+")|| !formulario.textField_leg_coment.getText().matches("[0-9]+"
53                  ||!formulario.textField_mod_tam_max.getText().matches("[0-9]+")|| !formulario.textField_mod_tamaño.getText().matches("[0-9]
54                  || !formulario.textField_uv_noinic.getText().matches("[0-9]+")|| !formulario.textField_uv_no_usada.getText().matches("[0-9]
55              JOptionPane.showMessageDialog(null,"Debe introducir un valor entero para la puntuación");
56
57          }
58          else{
59              //se guardan los valores actuales ingresados
60              formulario.guardarCampos();
61              //declaración del resto de clases de la capa Módulo con la lógica de la aplicación
62              Legibilidad validarLegib;
63              Modularidad validarModulo;
64              UsoVariables variables;
65              AccesoBBDD bbdd;//declaración del objeto de BBDD
66              String nombreArchivo="";
67              File directorio = new File(formulario.getRutaOrigen());
68              String[] listaDirectorio = directorio.list();
69              String mensajeTamañoModulos = "El tamaño máximo que debe tener un subprograma es: "+formulario.getTamañoMaxModulo();
70              //validaciones en las rutas especificadas
71              if (listaDirectorio == null){
72                  System.out.println("No hay ficheros en el directorio especificado");
73                  JOptionPane.showMessageDialog(null,"No hay ficheros en el directorio especificado");
74              }
75              else {
76                  Automatizacion archivo = new Automatizacion();
77                  //se guarda la sesion
78                  archivo.guardarSesion(ficheroSesion.getPath(),formulario.txtLenguaje.getText(), formulario.txtRutaBBDD.getText(), formulari
79                  for (int x=0;x<listaDirectorio.length;x++){
80                      nombreArchivo=listaDirectorio[x];
81                      try {
82                          bbdd=new AccesoBBDD();
```

Figura 19. Extracto de método main en clase Automatización (Controlador). Botón Aceptar

Se puede ver cómo al pulsar el botón ***Aceptar***:

- Se realizan las validaciones pertinentes para los campos introducidos (los campos obligatorios deben estar rellenos, los campos de pesos de los errores deben ser de tipo numérico y el lenguaje introducido debe ser válido de acuerdo a la BBDD)

- Se crean los objetos del resto de clases, tanto de la parte de Visión como de Base de datos.

- Se guarda la sesión actual para cargarla posteriormente en caso de necesitarla el usuario. Para el rol de profesor, guardará las últimas modificaciones en los pesos y, en caso de necesitar modificarlos en una nueva sesión, se habilitarán de nuevo introduciendo de nuevo el *Password* y pulsando **Modificar**.

- Una vez validados los campos, se hacen las llamadas a las clases que componen la capa Modulo (Legibilidad, Modularidad y UsoVariables), ver Figura 20.

*Automatizacion.java | Legibilidad.java | Modularidad.java | UsoVariables.java | Configuracion.java | AccesoBBDD.java

```
93      if(!esLenguaje)JOptionPane.showMessageDialog(null,"Ese lenguaje no existe en la BBDD. Debe modificar la Excel y añadi
94      //se crean los objetos del resto de clases
95      validarLegib= new Legibilidad(formulario.txtLenguaje.getText(), formulario.txtRutaBBDD.getText());
96      validarModulo = new Modularidad(formulario.txtLenguaje.getText(), formulario.txtRutaBBDD.getText());
97      variables = new UsoVariables(formulario.txtLenguaje.getText(), formulario.txtRutaBBDD.getText());
98      //llamadas al resto de modulos de la lógica de la aplicación
99      Vector<String> entrada = bbdd.leerTextoArchivo(formulario.getRutaOrigen(), nombreArchivo);
100     validarLegib.añadirTextoLegibilidad("Se va a analizar el archivo "+nombreArchivo);
101     validarLegib.añadirTextoLegibilidad("*****************************************\n*     ERRORES DE LEGIBILIDAD
102     validarLegib.añadirTextoLegibilidad("==== ERRORES DE INDENTACIÓN ====");
103     validarLegib.validarTabulacion(entrada);
104     validarLegib.añadirTextoLegibilidad("\n==== ERRORES DE COMENTARIOS ====");
105     validarLegib.validarComentariosAntes(entrada);
106     validarLegib.añadirTextoLegibilidad("\n==== ERRORES DE ESTILO ====");
107     validarLegib.validarEstiloMayusculas("Programa",validarLegib.obtenerNombresSubPrograma("PROGRAM", entrada));
108     validarLegib.validarEstiloMayusculas("Constante",validarLegib.obtenerConstantes(entrada));
109     validarLegib.validarEstiloMinusculas("Variable", validarLegib.obtenerVariables(entrada));
110     validarLegib.validarEstiloMinusculas("Función",validarLegib.obtenerNombresSubPrograma("FUNCTION", entrada));
111     validarLegib.validarEstiloMinusculas("Subrutina",validarLegib.obtenerNombresSubPrograma("SUBROUTINE", entrada));
112     validarLegib.añadirTextoLegibilidad("==== ERRORES DE SALTO INCONDICIONAL ====");
113     validarLegib.validarSaltoIncondicional(entrada);
114     validarLegib.añadirTextoLegibilidad("\n==== ERRORES DE DECLARACIÓN IMPLÍCITA DE VARIABLES ====");
115     validarLegib. validarDeclaracionImplicita(entrada);
116     validarLegib.añadirTextoLegibilidad("\n==== ERRORES DE FORMATO ====");
117     validarLegib.validarFormatoNombres(validarLegib.obtenerFormatoNombres(entrada),entrada);
118     bbdd.escribirProgramaArchivo(formulario.getRutaResultado(), "Documentación automática de "+nombreArchivo, validarLegi
119     int numeroErroresLegibilidad =validarLegib.getNumeroErroresLegibilidad();
120     //JOptionPane.showMessageDialog(null,"Número total de errores de Legibilidad: "+numeroErroresLegibilidad);
121     validarModulo.añadirTextoModularidad("*****************************************\n*     ERRORES DE MODULARIDAD
122     validarModulo.añadirTextoModularidad(mensajeTamañoModulos);
123     validarModulo.añadirTextoModularidad("==== ERRORES DE TAMAÑO DE SUBPROGRAMAS ====");
124     validarModulo.validarTamañoSubprogramas(formulario.getTamañoMaxModulo(),entrada);
125     validarModulo.añadirTextoModularidad("==== ERRORES DE VALORES DE RETORNO DE SUBPROGRAMAS ====");
126     validarModulo.añadirTextoModularidad("Subrutinas:");
127     validarModulo.validarValorRetornoSubrutinas (entrada);
128     validarModulo.añadirTextoModularidad("Funciones:");
129     validarModulo.validarValorRetornoFunciones (entrada);
130     validarModulo.añadirTextoModularidad("==== ERRORES DE ARGUMENTOS DE SUBPROGRAMAS ====");
131     validarModulo.añadirTextoModularidad("Subrutinas:");
132     validarModulo.validarArgumentosSubrutina(entrada);
133     validarModulo.añadirTextoModularidad("Funciones:");
134     validarModulo.validarArgumentosFuncion(entrada);
135     validarModulo.añadirTextoModularidad("==== ERRORES DE INTERFAZ ====");
136     validarModulo.validarInterfazFuncion(entrada);
137     validarModulo.validarEscrituraAntesLectura(entrada);
138     int numeroErroresModularidad=validarModulo.getNumeroErroresModularidad();
139     //JOptionPane.showMessageDialog(null,"Número total de errores de Modularidad: "+numeroErroresModularidad);
140     variables.añadirTextoUsoVariables("*****************************************\n*     ERRORES DE USO DE VARIABLES
141     variables.validarUsoVariablesDeclaradas(entrada);
142     //revisar porque da nullpointer exception-- corregido!
143     variables.validarVariableSinInicializar(entrada, formulario.getRutaOrigen());
144     variables.validarAsignacionVariableDoble(entrada);
145     variables.validarVariableGlobal(entrada);
146     variables.validarDobleReferencia(entrada);
147     int numeroErroresVar=variables.getNumeroErroresUsoVariables();
148     variables.añadirTextoUsoVariables("*  FIN DE CORRECCIÓN DE "+nombreArchivo+"  *");
```

Figura 20. Llamadas desde el Automatizador (Controlador) a la capa Modulo con la lógica

Del mismo modo, al pulsar el botón Modificar*:*

- Se valida si es o no correcta la contraseña introducida, en caso afirmativo, se activan el resto de campos de la interfaz, ver Figura 21.

```
btnModificar.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {
        if( formulario.getPasswordField().getText().equals("automatizador") ) {
            //habilitar los campos con el botón modificar
            formulario.activarCampos();
        }
        else{
              JOptionPane.showMessageDialog(null,"No tiene permiso para cambiar los valores de configuración");
        }
    }
});
```

Figura 21. Acciones al pulsar botón "Modificar"

4.4. Diseño del Modelo

La capa del modelo contiene la lógica del sistema y está formada por las tres clases, pertenecientes a las tres líneas de requisitos funcionales (Legibilidad, Modularidad y UsoVariables) y además se establece la conexión con la base de datos de Excel.

4.4.1. Diseño de Legibilidad

Para el diseño de la lógica del sistema referente a la legibilidad del código se han tomado las decisiones necesarias para cumplir los requisitos especificados, detallados a continuación a nivel de diseño, ver capítulo 3 con los Requisitos de la Herramienta:

1. **Indentación** (requisito RF-05). *El código debe estar correctamente indentado, según la profundidad de cada instrucción dentro del conjunto de bloques, ya sea mediante espacios en blanco o tabuladores.*

   Para establecer un valor de indentación, se ha tomado como línea para patrón de indentación, la primera línea no comentada que incluya un nivel de indentación distinto de 0.

   La herramienta lo resuelve mostrando la línea que ha tomado como patrón de indentación, muestra el número de espacios y/o tabulaciones aplicados en esa línea y, a continuación, debe replicarse ese patrón.

   De modo que, por ejemplo, si se encuentra la primera sangría con 3 espacios, va a calcular múltiplos de 3 espacios por cada nivel de indentación, ver Figura 24.

   En caso de que la línea en donde se detecta la primera sangría fuera excesivamente pequeña o grande, añade un mensaje avisando del problema. Ver Figura 22 y 23.

```
La sangría elegida aplicable es la de su primera aparición en el código, en la fila: 4
AVISO: ************La sangría aplicada es muy pequeña (1 espacio)*********
```

Figura 22. Aviso de mínima indentación

```
La sangría elegida aplicable es la de su primera aparición en el código, en la fila: 4
AVISO: ************La sangría aplicada es muy grande ( 2 ) tabulaciones*********
```

Figura 23. Aviso de demasiada indentación.

Se **añade** un nivel de indentación después de las líneas que contienen:

- El inicio de Sentencias de Control (por ejemplo, se incluye para Fortran: IF, ELSE IF, SELECT CASE, CASE, ELSE).
- El inicio de un procedimiento o subproceso (por ejemplo en Fortran: PROGRAM, FUNCTION, SUBROUTINE).

Se **disminuye** un nivel de indentación a partir de la línea que contiene:

- Alternativas tras otra sentencias de Control (por ejemplo, se incluye para Fortran: ELSE IF, CASE, ELSE).
- Final de sentencias de control, procedimientos y subprocedimientos
- Sentencias de control repetitivas o bucles

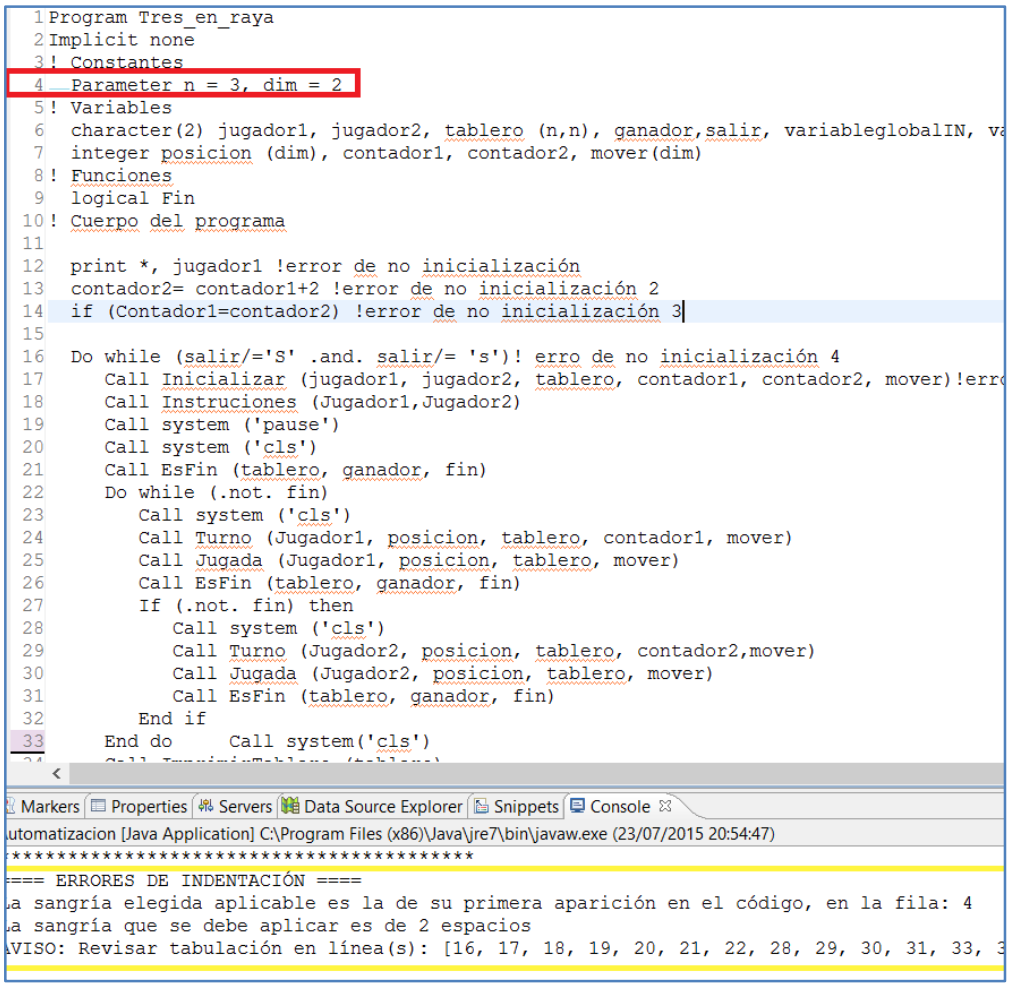


Figura 24. Ejemplo Indentación. Línea que contiene sangría a aplicar y resultado

2. **Comentarios** (requisito RF-06). *El código debe estar comentado. Se deben incluir comentarios al principio del programa y antes de cada subrutina. Dichos comentarios se utilizan para generar una documentación automática que describa a fondo el programa desarrollado y cada una de los subprogramas que lo forman.*

   Para resolver esto la herramienta recorre las líneas desde el inicio de un programa o subprograma hacia arriba comprobando si existen líneas que comiencen por los

valores ingresados en la BBDD como de inicio de comentario, ver Figura 25.

```
==== ERRORES DE COMENTARIOS ====
ERROR: Noy hay comentarios para el código que lo exige en las líneas: [0, 66, 121, 159, 259, 282, 3
```

Figura 25. Error de comentario

3. **Estilo** (requisito RF-07). *Se debe validar que los nombres de variables, constantes, procedimientos y funciones deben seguir un formato determinado a lo largo de todo el programa.*

   Se validará que las constantes comiencen por letra mayúscula y las variables, variables, procedimientos o subprocedimientos (funciones y subrutinas) con la primera letra en minúsculas, ver Figura 26.

```
==== ERRORES DE ESTILO ====
ERROR: Constante con nombre: n empieza en minúsculas en la linea: 4
ERROR: Constante con nombre: dim empieza en minúsculas en la linea: 4
ERROR: Constante con nombre: dim empieza en minúsculas en la linea: 50
ERROR: Constante con nombre: n empieza en minúsculas en la linea: 70
ERROR: Constante con nombre: dim empieza en minúsculas en la linea: 70
ERROR: Constante con nombre: n empieza en minúsculas en la linea: 96
ERROR: Constante con nombre: dim empieza en minúsculas en la linea: 96
ERROR: Constante con nombre: n empieza en minúsculas en la linea: 125
```

Figura 26. Errores de Estilo

4. **Salto Incondicional** (requisito RF-08). *No se permite el uso de sentencias de salto incondicional ni ruptura del flujo de ejecución.*

   La herramienta simplemente busca la sentencia asociada a un salto incondicional en cualquier lugar del programa y muestra el error junto con la línea. En la figura 27 se pone un ejemplo de error causado por incluir en código en lenguaje Fortran la sentencia “GO TO”.

```
==== ERRORES DE SALTO INCONDICIONAL ====
ERROR: No se permite el uso de sentencias de salto incondicional, en línea: 51
```

Figura 27. Error de salto incondicional

5. **Declaración Implícita** (requisito RF-09). *No está permitido la definición implícita de variables.*

   O lo que es lo mismo, debe usarse declaración explícita de variables. Pues, si no declaro las variables, los tipos de datos se asignan por defecto (por ejemplo, en Fortran, las variables que comienzan por “i”, “j”, “k”, “l”, “m” o “n” se asocia al tipo entero, el resto se consideran de tipo real). Esto puede generar errores muy difíciles de detectar.

   La herramienta verifica que al comienzo de un procedimiento aparezca la sentencia asignada para impedir la asignación automática de variables, en el caso del lenguaje Fortran, la sentencia es “IMPLICIT NONE” que permite que aparezcan errores de

compilación por la no declaración de variables.

Aparece el error en caso de no encontrar esta sentencia como primera línea (que no esté vacía o comentada) después de la línea de comienzo del procedimiento.

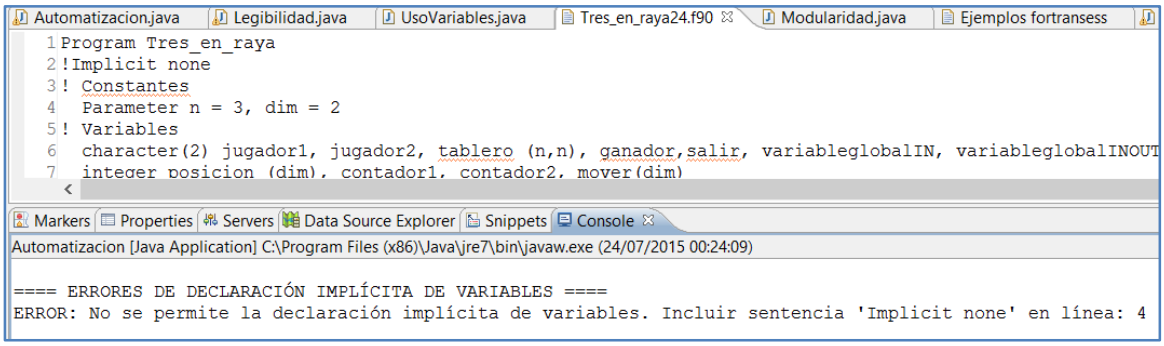


Figura 28. Error por no incluir la sentencia de eliminar las variables implícitas.

6. **Formato** (requisito RF-10). *El estilo elegido de nombrar las variables, constantes, nombres de procedimientos y subprocedimientos debe ser aplicado uniformemente a lo largo de todo el programa.*

Esto hace alusión a la manera de nombrar los identificadores, no se debe mezclar, por ejemplo estos formatos en un programa: *variable_uno* y *variableUno*. Se deben nombrar todas con igual formato.

Primeramente, debemos elegir el formato patrón para aplicar la comparación con el resto del código a corregir. Se ha decidido que primero se tome el formato asignado al nombre de la primera función en aparecer, en su defecto, será la primera subrutina. Ver ejemplo en figura 29.

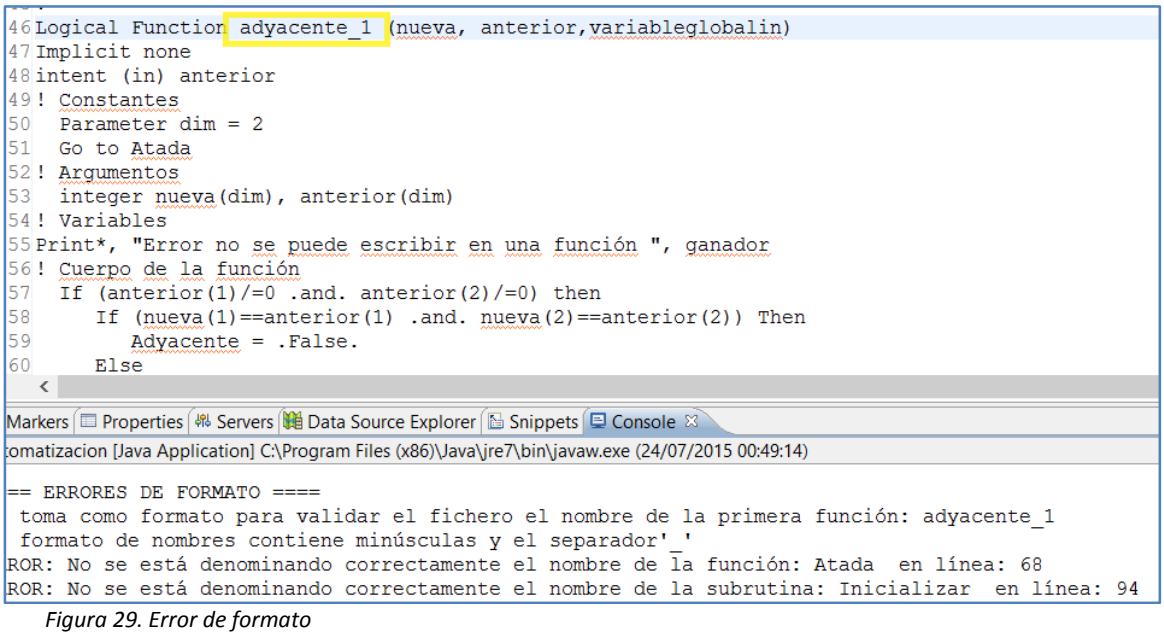


Figura 29. Error de formato

4.4.2. Diseño de Modularidad

Para el diseño de la lógica del sistema referente a la modularidad del código se han tomado las decisiones necesarias para cumplir los requisitos especificados, detallados y desarrollados a continuación a nivel de diseño, ver capítulo 3 con los Requisitos de la Herramienta:

1. **Tamaño módulo y tamaño máximo módulo** (requisitos RF-11 y RNF-09 respectivamente). *El programa debe estar lo suficientemente modulado. Una de las formas de controlar esto es restringiendo el tamaño de los subprogramas.*

   Es importante analizar el tamaño de las unidades de código (métodos o subprocesos) que componen el sistema. Un gran volumen del tamaño del sistema no significa que necesariamente haya problemas en el diseño (simplemente puede tener mucha funcionalidad implementada).

   Sin embargo, detectar unidades individuales excesivamente grandes sí puede indicar un mal diseño.

   Los módulos son considerados, según su tamaño como [MIE]:

   - Muy grande → nº de líneas de código por módulo: > 100
   - Grande → nº de líneas de código por unidad: (50-100]
   - Medio → nº de líneas de código por unidad: (10-50]

   Se toma de la capa de la interfaz el valor máximo de líneas aceptado en un subprocedimiento (campo Tamaño max. Módulo), por defecto el valor está fijado en 100 líneas y determina el tamaño máximo que pueden tener todos módulos (métodos y subprocedimientos).

   La herramienta lee las cabeceras de los subprogramas y va contando líneas hasta llegar al fin de dichos subprogramas. Ver Figura 30.

```
*****************************************
El tamaño máximo que debe tener un subprograma es: 20
==== ERRORES DE TAMAÑO DE SUBPROGRAMAS ====
ERROR: Se está superando el tamaño máximo requerido (de 20 líneas) para la subrutina: Inicializar es de: 24
ERROR: Se está superando el tamaño máximo requerido (de 20 líneas) para la subrutina: Instruciones es de: 30
ERROR: Se está superando el tamaño máximo requerido (de 20 líneas) para la subrutina: EsFin es de: 95
```

Figura 30. Error en tamaño de subprogramas

2. **Valor retorno** (requisito RF-12*). El programa principal coordina el flujo de ejecución, delegando tareas en los respectivos subprogramas. Se fuerza el uso correcto de subprogramas, es decir, el estudiante tiene que distinguir entre funciones y procedimientos. Aquellos subprogramas que devuelvan un solo valor serán funciones y el resto serán procedimientos.*

   a. *Toda función debe tener un valor de retorno.*
   b. *Las subrutinas no pueden tener valor de retorno*

   La herramienta muestra error en las funciones en caso de que no tengan valor de retorno, o tengan un valor de retorno diferente a los tipos de datos existentes. Comprueba también que, al menos una instrucción dentro de la función, contenga

una asignación del nombre de la función, ver Figura 31.

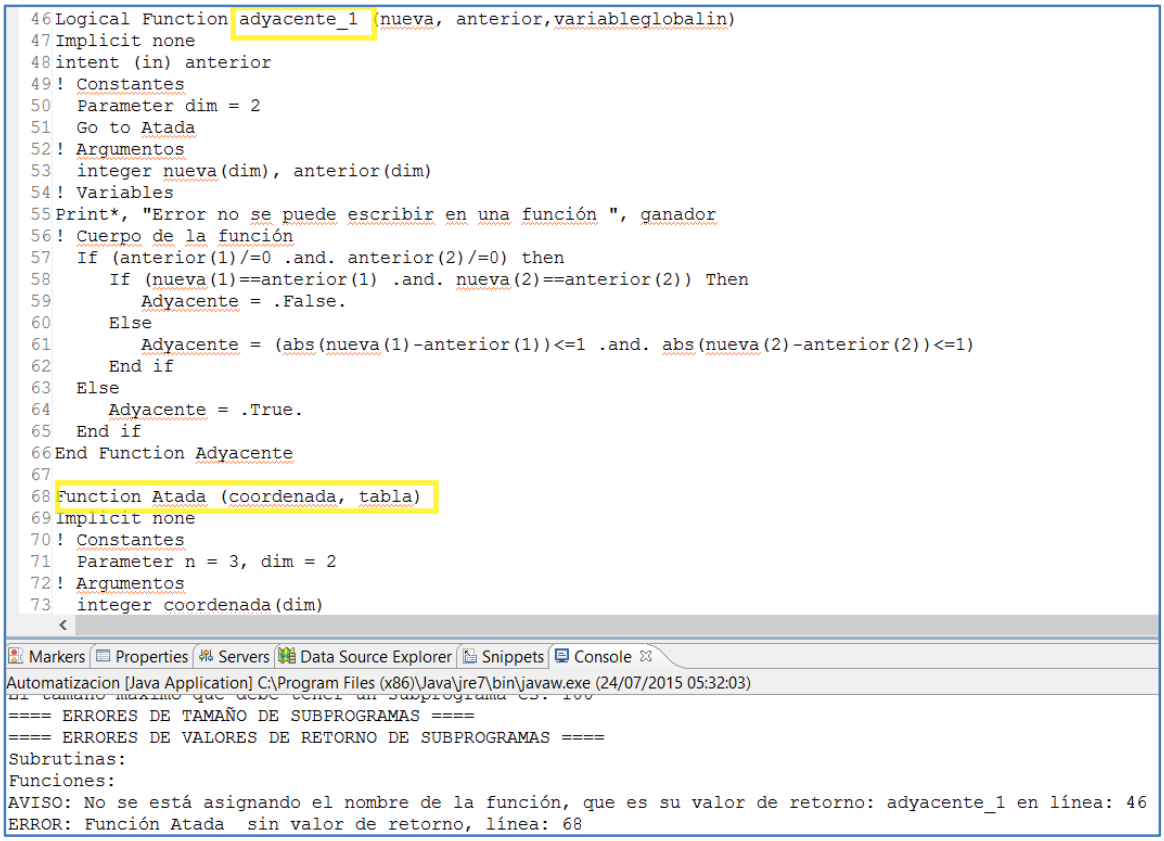


Figura 31. Ejemplo de errores de no retorno de funciones

Del mismo modo, comprueba que las subrutinas no tengan ningún valor de retorno, sea tipo de dato existente o no, ver Figura 32.

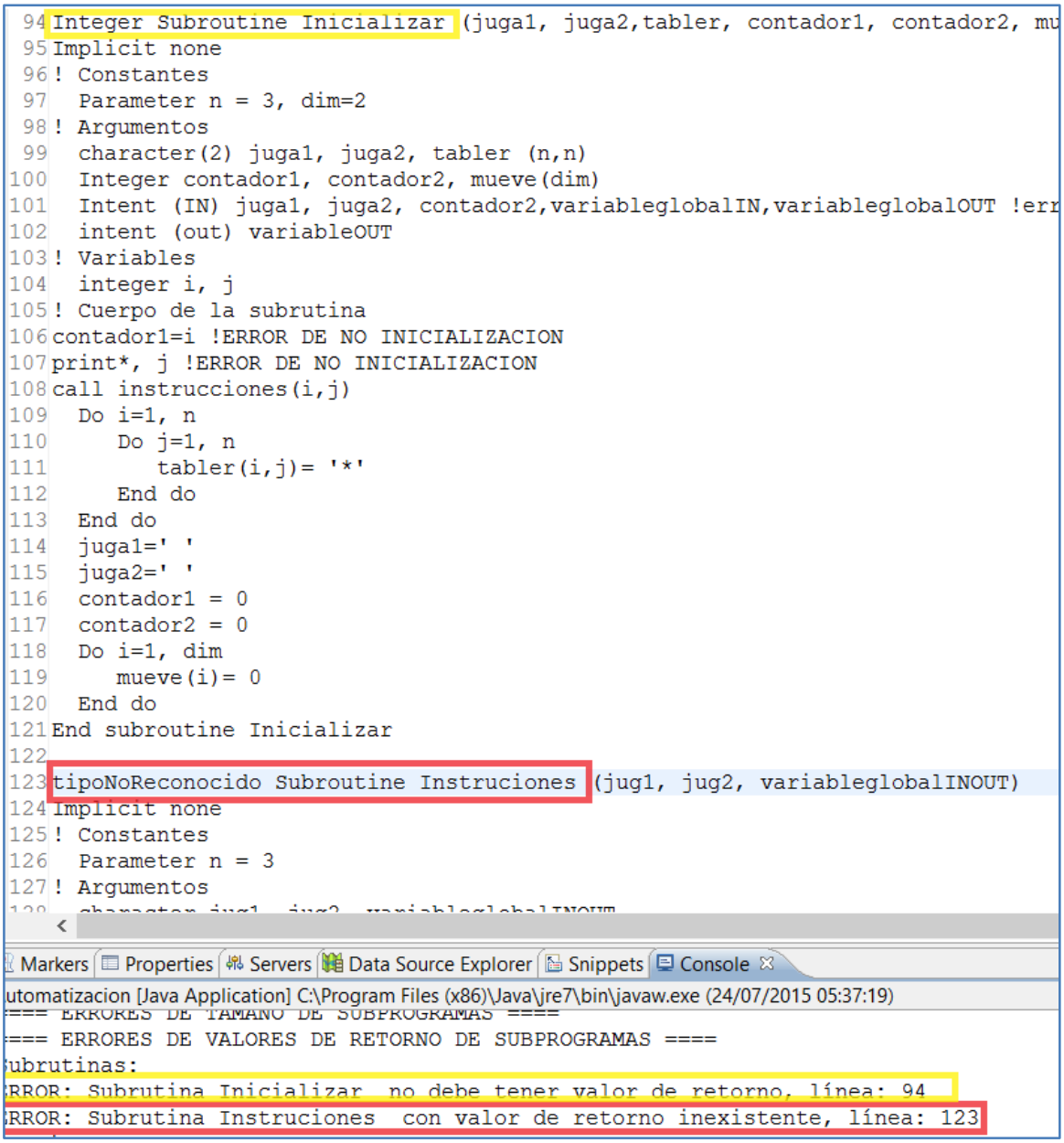


Figura 32. Ejemplo de error de valor de retorno de subrutinas

3. **Uso Argumentos (Valor/Referencia)** (requisito RF-13). *Los argumentos de una subrutina pueden ser pasados por valor o por referencia, en función de si se va a querer modificar su contenido dentro de la subrutina o no. Se controlan los siguientes aspectos:*
    a. *No se debe modificar una variable pasada por valor.*
    b. *Los argumentos pasados por referencia deben ser modificados.*
    c. *Impedir el paso de argumentos con doble referencia.*

    Para resolver esto, la herramienta:

    - Valida que los argumentos de una subrutina (pasados por referencia o por valor) sean modificados o no, respectivamente, pudiendo o no ser utilizados para leer y escribir.

- Valida que los argumentos de una función (pasados por valor) no son modificados y no pueden ser utilizados para leer y escribir.
- Los argumentos pasados con doble referencia se trata en el módulo de Uso de variables, más adelante.

Para distinguir si un argumento es pasado por valor o por referencia, se facilita a los alumnos la siguiente documentación en código Fortran, utilizando la palabra reservada INTENT y a continuación su uso:

- ✓ argumentos de entrada: su valor no deberá modificarse durante la ejecución del subprograma.

  tipo argumento

  INTENT (IN) argumento

- ✓ argumentos de salida: su valor se modificará durante la ejecución del subprograma.

  tipo argumento

  INTENT (OUT) argumento

- ✓ argumentos de entrada y salida

  tipo argumento

  INTENT (INOUT) argumento

Si no se utiliza la sentencia INTENT, por defecto considerará que el argumento es de tipo INTENT(INOUT).

La herramienta, por tanto, hace las siguientes validaciones de argumentos, ver Figura 33:

- Los argumentos de tipo OUT se deben modificar (ya sea en una sentencia de asignación (antes del =) o en una sentencia de entrada “read”)
- Los argumentos de tipo INOUT se deben modificar (ya sea en una asignación (antes del =) o en una sentencia de entrada “read”) y además, deben ser usados (ya sea en como expresión de una sentencia de asignación, o en una sentencia de control)
- Los argumentos de tipo IN no deber ser modificados.

Según lo anterior, también se valida que, en Fortran:

- Las subrutinas pueden tener declaraciones de INTENT (IN), INTENT (OUT) e INTENT(INOUT)
- Las funciones sólo pueden tener declaraciones de INTENT (IN).
- Adicionalmente, se incluyen validaciones de formato en los argumentos, esto es, que avise en caso de modificaciones del identificador con respecto a la declaración de la variable o argumento. Por ejemplo, si se ha declarado con todas las letras en minúsculas, así debería mantenerse a lo largo del

programa, ver Figura 33.

En caso de lenguaje Java, todos los argumentos son considerados pasados por valor.

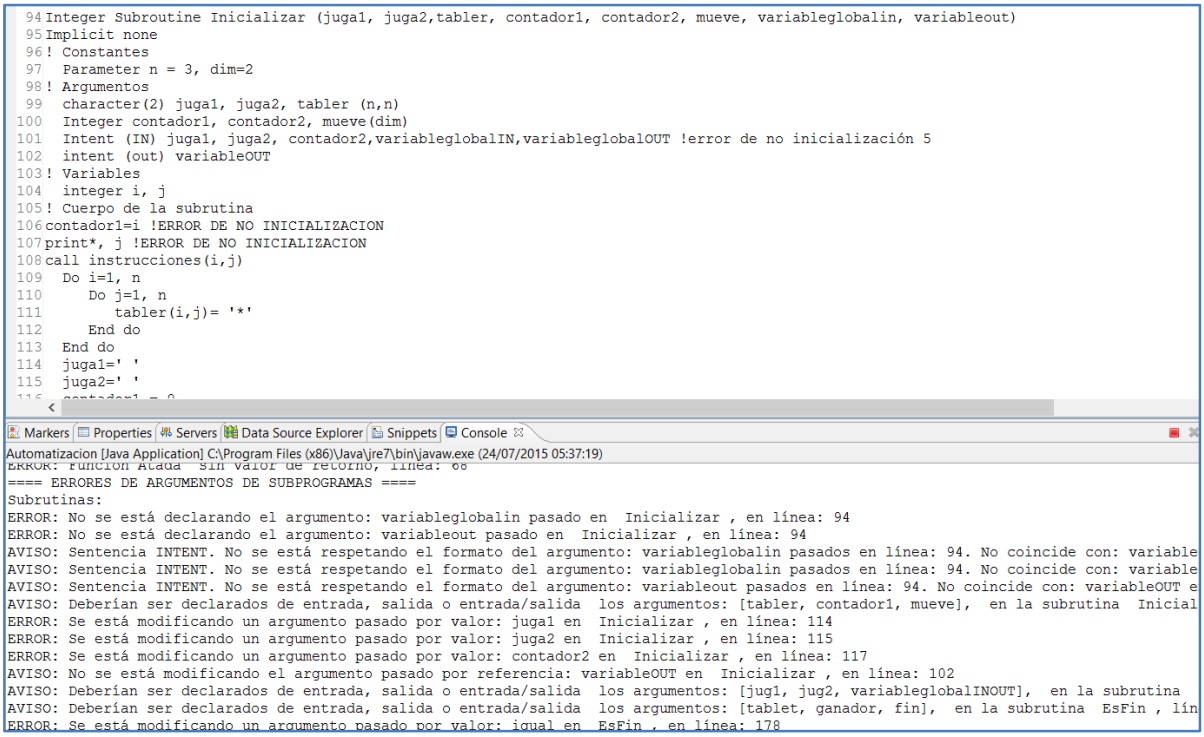


Figura 33. Ejemplo de errores de uso de argumentos por valor o referencia

4. **Declaración de argumentos** (requisito RF-14). Este requisito surge posteriormente, fuera del planteamiento inicial, debido a la problemática anterior de distinguir cuándo un argumento está pasado por valor o por referencia. Entonces surge la opción de distinguir los argumentos con declaraciones de tipo INTENT (IN, INOUT y OUT). De este modo se facilita la validación:

   - En Subrutinas: Se pueden declarar los argumentos de cualquier tipo: INTENT (IN), INTENT (OUT) e INTENT(INOUT) dado que pueden ser argumentos pasador por valor o por referencia indistintamente, aquí no hay restricción.
   - En Funciones: Solo se puede declarar los argumentos de tipo INTENT (IN). Por lo que es las funciones donde se realizan las validaciones, ver Figura 34:
     - Que todos los argumentos pasados en una función sean declarados de tipo IN (de entrada). Muestra mensaje de que deben declararse todas de tipo IN.
     - En caso de que no se declaren, se tomarán como INOUT y se tratarán como tal.
     - Que se mantenga el formato de escritura de los argumentos.

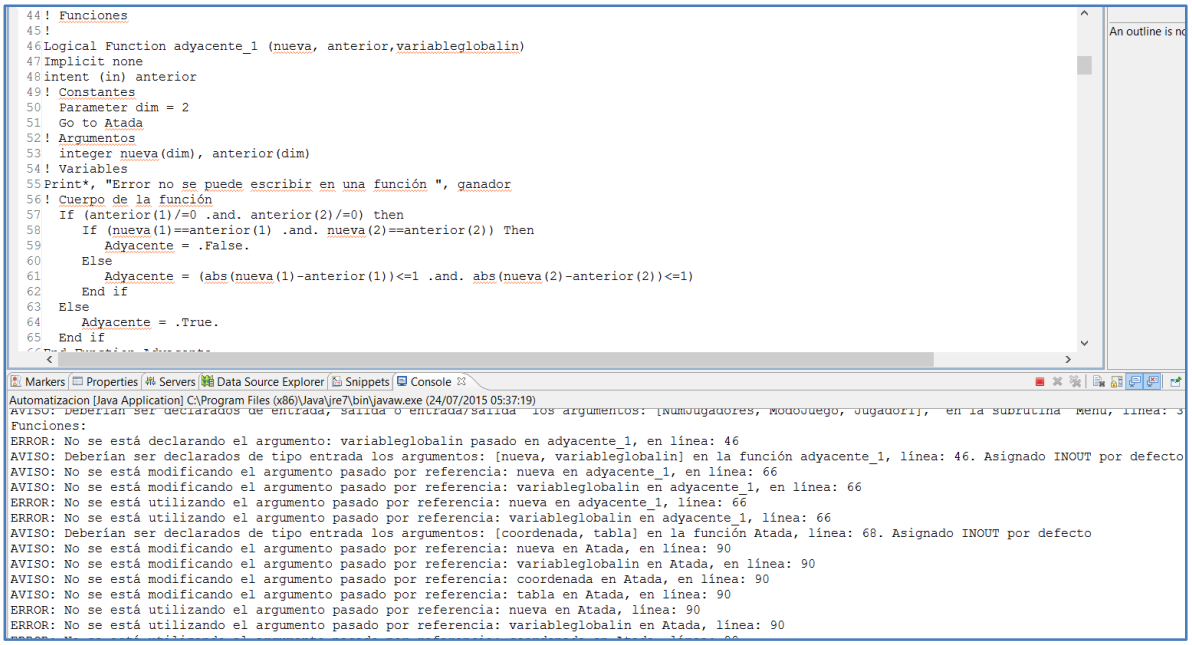


Figura 34. Ejemplo de errores de declaración de argumentos en función

5. **Interfaz** (requisito RF-15*). Relacionados con la interfaz de usuario, se controlan dos aspectos básicos.*

   a. *No se debe mezclar la lógica del programa con la interfaz de usuario. Esto es, un subprograma que calcule o modifique un dato no debe imprimir por pantalla, y viceversa. De este modo se aíslan las partes del programa que son más propensas a sufrir posterior mantenimiento (interfaz) de las que son más estables (funcionalidad), reduciendo los costes de mantenimiento [LOR].*
   b. *Siempre que se pida un dato por teclado, debe haber un mensaje previo de petición del dato.*
   c. *Siempre que se saque un dato por pantalla debe estar acompañado o precedido de un rótulo.*

   La herramienta realiza dos validaciones:

   - Comprueba que en una función no haya ninguna instrucción de lectura, ni escritura o impresión por pantalla.
   - En caso de lectura, comprueba que antes de leer un dato lo pida por pantalla.
     - En este caso comprueba la línea (no comentada ni vacía) inmediatamente antes de la de lectura, por lo que puede que esté en el código dentro de una sentencia de control. De ahí que muestre un AVISO en lugar de un ERROR.

   En caso contrario, añade el error a los resultados indicando dónde se encuentran los errores de interfaz. Ver Figura 35.

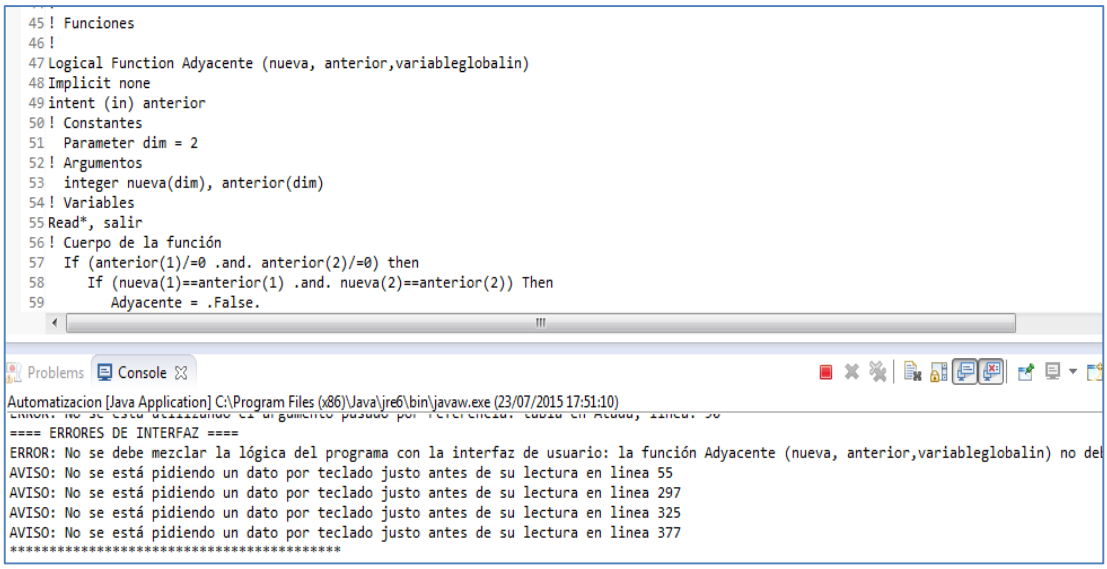


Figura 35. Error de interfaz en una función

4.4.3. Diseño de Uso de Variables

Para el diseño de la parte relativa al uso de variables y manejo de variables, se controla que la utilización que se hace de ellas es la correcta. La herramienta avisa, por tanto, en caso de encontrar las siguientes situaciones en el programa del alumno:

1. **No inicializada** (requisito RF-16*). Utilización del valor de una variable no inicializada.*

La herramienta identifica el error si las variables declaradas no han sido inicializadas, se validará que antes de usarse la variable, se ha producido uno de los siguientes casos:

- Se inicializan cuando en la variable (x):
    1. Hay una sentencia de asignación (x=...)
    2. Hay una instrucción de lectura (Por ejemplo: READ*, x)
    3. Se produce una llamada en la que la variable es un argumento pasado por referencia (Por ejemplo: CALL (argumentos pasados por referencia, es decir, modificables (tipo INTENT INOUT o OUT))
- Si no han sido inicializadas (de cualquiera de las opciones anteriores) y se produce uno de los siguientes casos, se muestra ERROR:
    1. La variable está dentro de una expresión de una asignación (por ejemplo, =...X), ver Figura 36.
    2. La variable está en una instrucción de escritura (Por ejemplo, PRINT*, x *)
    3. Se produce una llamada en la que la variable es un argumento pasado

por valor, es decir, no modificables (En Fortran, tipo INOUT o IN, en Java se consideran todas INOUT)

4. La variable está dentro de una sentencia de control alternativa, no siendo inicializada en su declaración. (Por ejemplo, IF (...x...) o DO WHILE(...x...))

5. La variable es un argumento pasado por referencia de un subprograma. Por ejemplo, en Fortran: FUNCTION (x,..) los argumentos de una función siempre son de entrada INTENT IN (NO MODIFICABLES), este caso se daría si la variable es global, por lo que también sacaría error de acceso a una variable global.

6. La variable es un argumento pasado por valor de un subprograma. Por ejemplo, en Fortran: SUBROUTINE (...x...) y el argumento es de tipo INTENT IN (NO MODIFICABLES), este caso se daría si la variable es global, por lo que también sacaría error de acceso a una variable global.

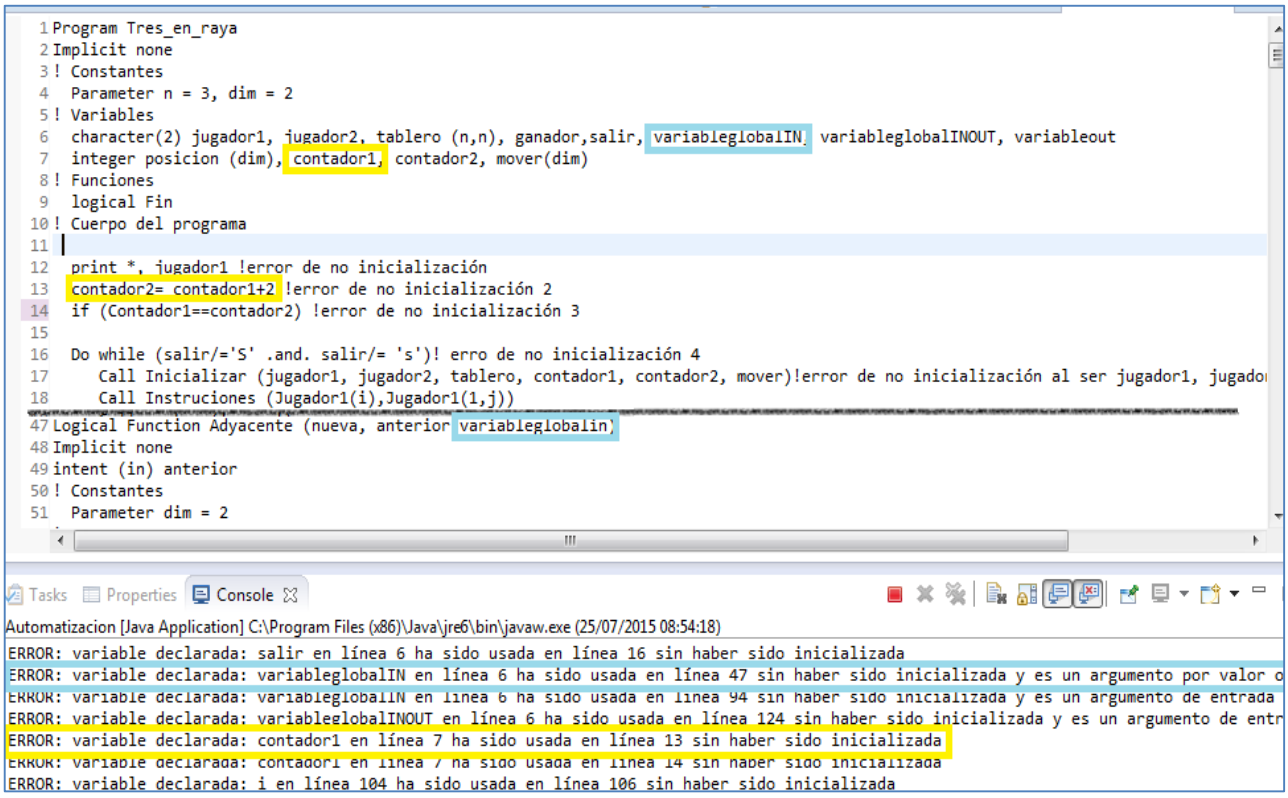


Figura 36. Errores de no inicialización de variables.

2. **No uso** (requisito RF-17). Declaración de una variable que no se utiliza nunca.

Se avisa al usuario de que la variable declarada no se está utilizando dentro del módulo donde se declaró. Ver Figura 37.

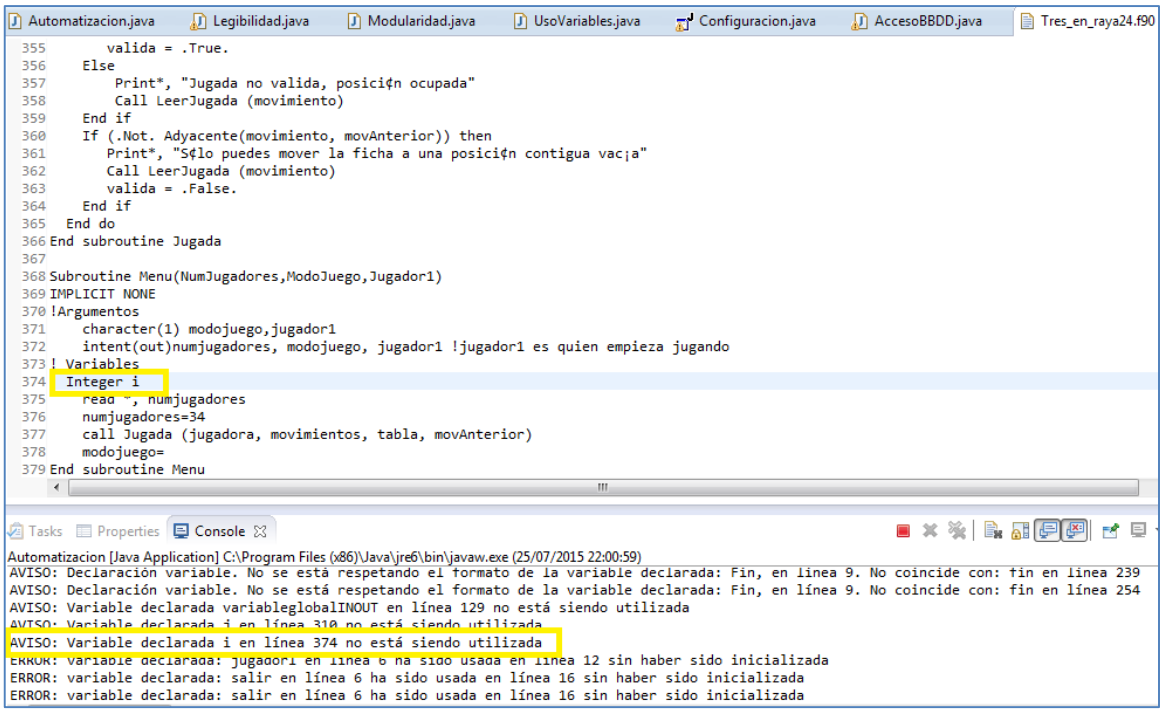


Figura 37. Error de variable declarada no usada

3. **Asignación doble** (requisito RF-18). Debe existir, al menos un uso por referencia entre dos asignaciones consecutivas a una variable.
   Comprobamos que la variable no esté siendo utilizada en una sentencia de control de cualquier tipo, ver Figura 38, donde se modifica la variable tras lo que finaliza el bucle y se reasigna la variable, esto sería correcto.

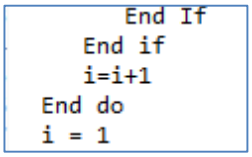


Figura 38. Situación de asignaciones dobles permitidas.

También se tiene en cuenta si la variable es una matriz, en ese caso si los índices son distintos entre asignaciones no detecta doble asignación.

La herramienta detecta, si alguno de los índices de la matriz es una variable y los guarda para comprobar si son modificados en las líneas siguientes, ver Figura 39.

Alerta de la modificación de la variable y añade el error, porque puede que se haya modificado al valor de la primera asignación.

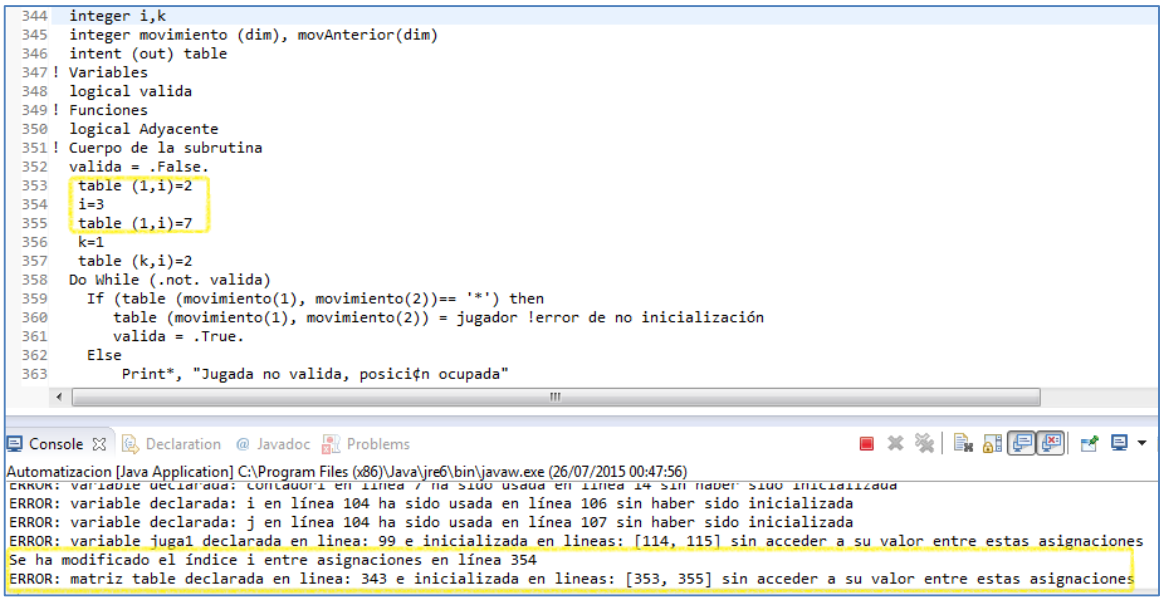


Figura 39. Error de doble asignación, alertando de modificación de índice

En el caso de las matrices, no se detectará una asignación de una variable nueva entre ambas asignaciones, aunque sean del mismo valor, ya que la herramienta sólo guarda las variables contenidas en la primera asignación, ver Figura 40.

```
table (1,n)=2
k=1
table (k,n)=2
```

Figura 40. La herramienta no detecta este tipo de asignación doble

La herramienta genera un error de asignación sobre una variable asignada anteriormente, sin hacer uso del valor entre ambas asignaciones. Ver Figura 41.

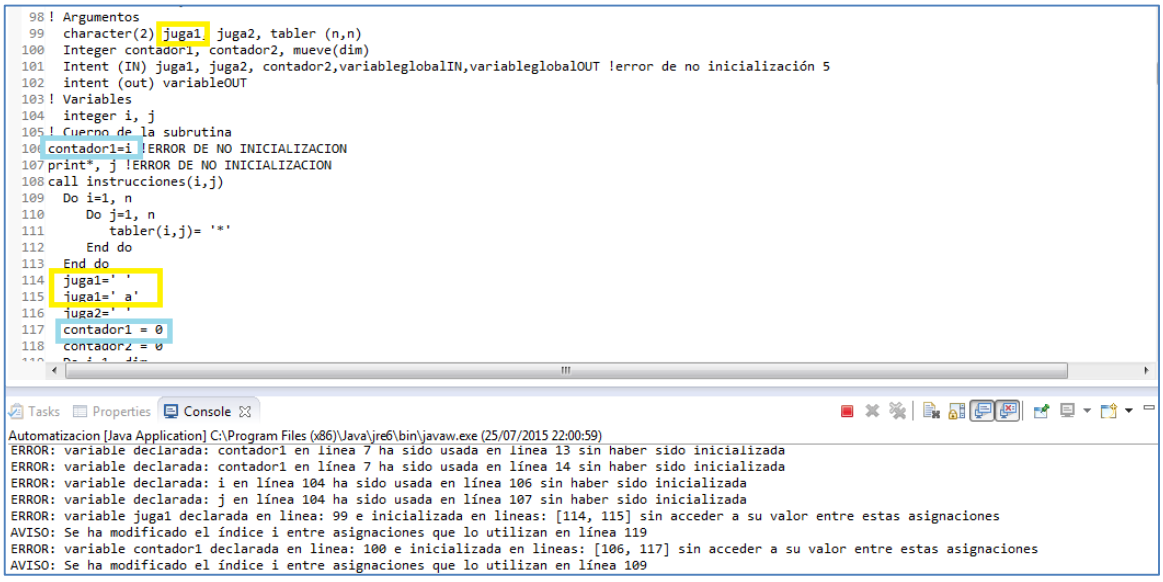


Figura 41. Errores de doble asignación sin modificar variable entre ambas

4. **Acceso a var**. Global (requisito RF-19). Acceso a variables globales por visibilidad [CCE+].

   Haciendo uso de las buenas prácticas en programación, no se debe acceder a variables globales, porque generaría errores no gestionados, difíciles de detectar posteriormente, constituyendo una de las principales causa de fracaso.

   La herramienta también detecta acceso a variable global aunque la variable se nombre igual dentro de un subproceso o método.

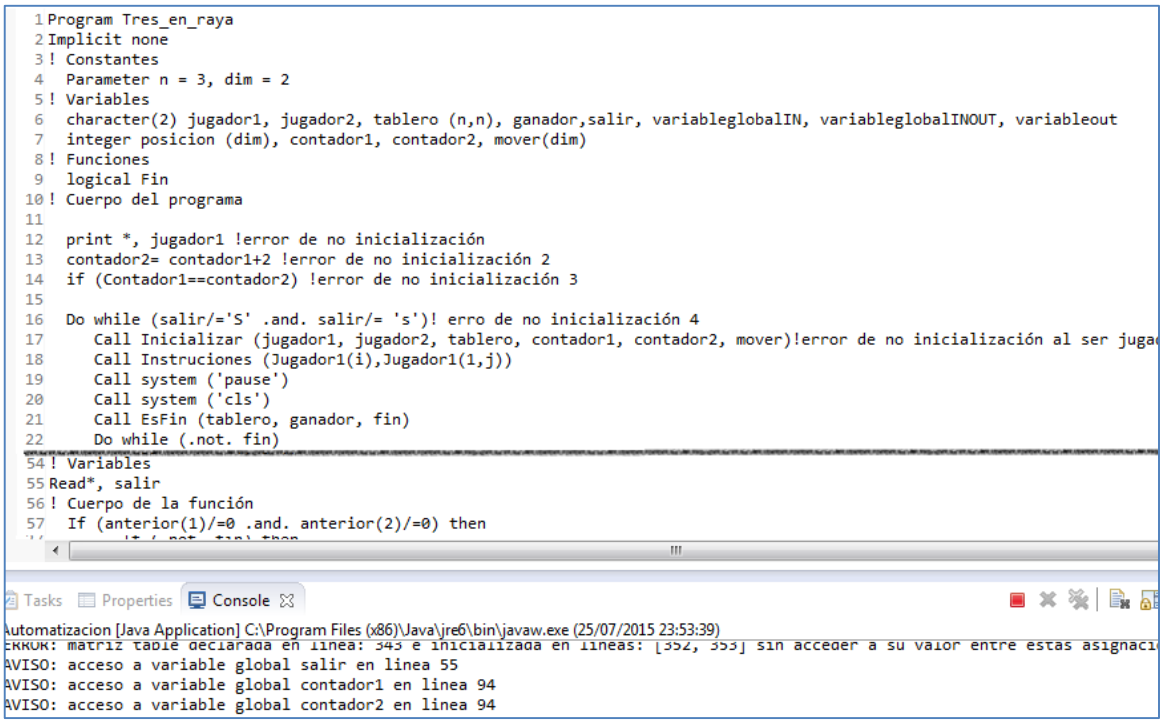


Figura 42. Aviso de acceso a variable global

5. **Doble referencia** (requisito RF-20). *Se impide el paso de argumentos con doble referencia, es decir, pasar por referencia más de una vez simultáneamente el mismo parámetro actual* [CCE+] *(se evitan efectos laterales).*

   - La doble referencia se da cuando en una llamada a un subprograma se utiliza más de una vez la misma variable. Por ejemplo: Subroutine SumaProducto(x, y, s, p) donde en “s” se calcula la suma de x e y, y en “p” el producto de x e y.

   - La llamada call SumaProducto (5,4, z, z) es una doble referencia. Si es verdad que no es conveniente que los argumentos se llamen igual que las variables que se usarán en las llamadas aunque eso no se llame doble referencia.

   - No hay problema en lo siguiente: subroutine suma (x,y,z)
     subroutine producto (x,y,z)

Método que saca un aviso en caso de que en una llamada estemos pasando el mismo argumento, aunque sea con índices diferentes (en caso de ser una matriz), ver figura 43.

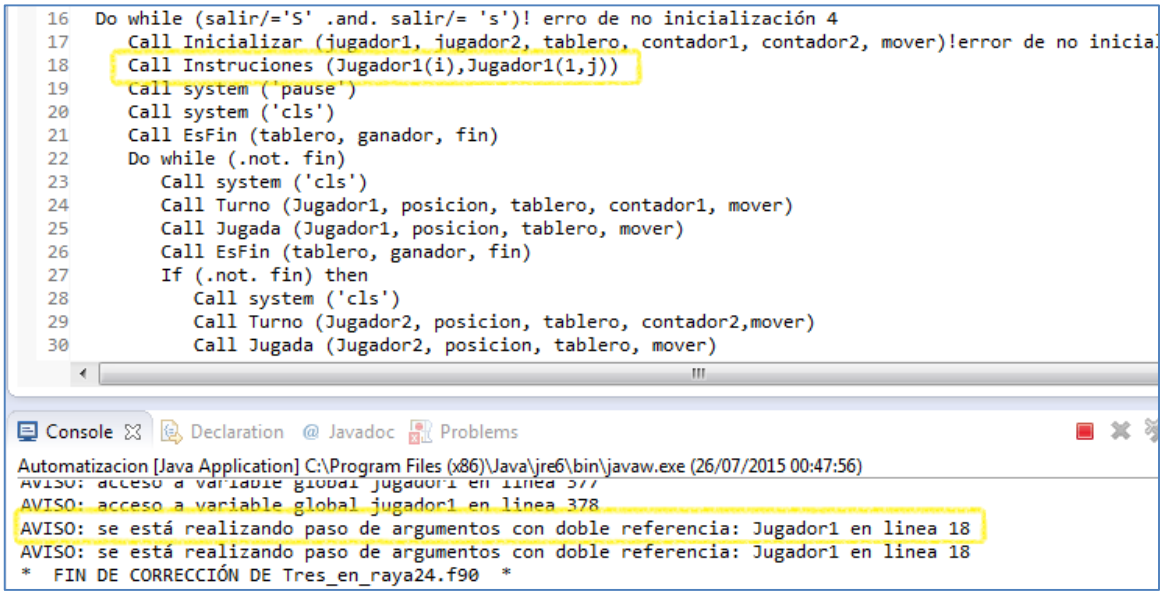


Figura 43. Error de doble referencia

Adicionalmente a todas las anteriores, valida que se respeta el formato del identificador de la variable, es decir, alerta del empleo de mayúsculas o minúsculas en el uso de la variable que difieran con el formato utilizado en la declaración de la misma. Ver figura 44. Estos errores pueden ser detectados en el propio editor antes de la compilación.

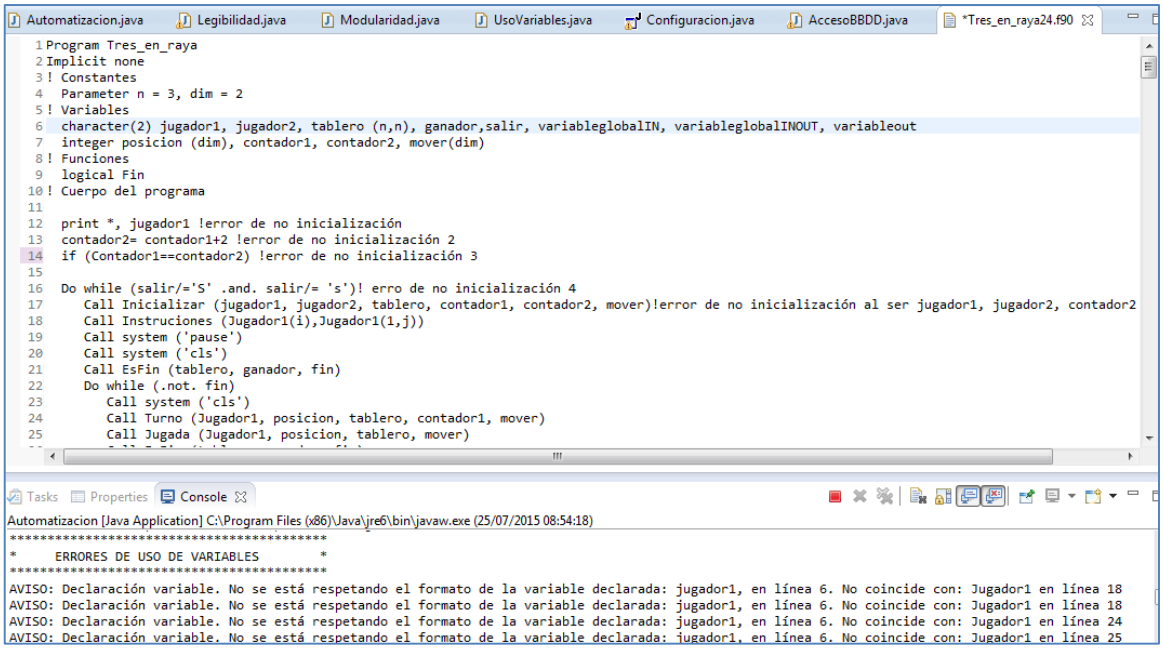


Figura 44. Ejemplo de errores de formato de las variables tras su declaración

4.4.4. Interacción con la Base de Datos

Desde todas las clases de la capa Modelo se accede a la base de datos, es decir, se llama a la clase AccesoBBDD.

Como puede verse se hace una llamada al constructor pasándole el lenguaje y la ruta del fichero Excel con la BBDD y se establece conexión con la BBDD, ver Figura 45.
Para cada módulo de la lógica (Legibilidad, Modularidad y UsoVariables) se necesitarán unas columnas u otras (con sus sentencias, tipos e instrucciones) dependiendo de la necesidad de cada una de ellas.

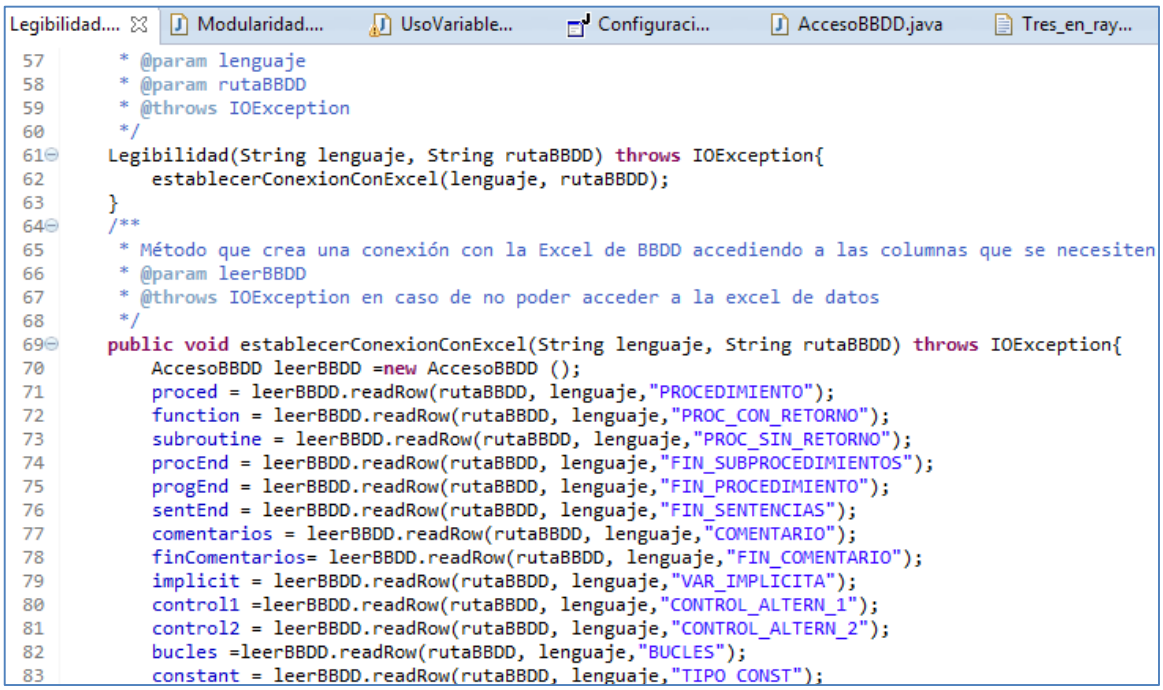


Figura 45. Acceso a la BBDD desde la capa Modelo

4.5. Diseño de la Base de Datos

Dada la configurabilidad de la herramienta, los anteriores aspectos se pueden aplicar a otros los lenguajes de programación (requisito RF-01) de las diferentes asignaturas.

Para esto, se ha diseñado la base de datos de lenguajes en Excel, así permite mantener de manera sencilla los datos de la herramienta.

El documento Excel será precargado por el usuario en la ruta especificada en la interfaz, ver Figura 46.

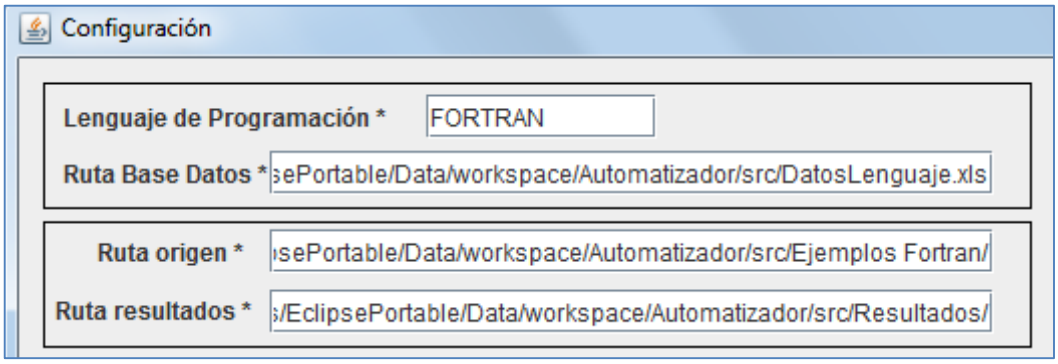


Figura 46. Excel de datos para lenguaje Fortran

En caso de que el lenguaje introducido no esté cargado previamente, ver Figura 47. La interfaz hace una lectura de los nombres de las Hojas de la Excel identificando dicho

nombre con un lenguaje de programación. Por lo que si al introducir en el campo Lenguaje de la interfaz un nombre nos da un mensaje de error como en las Figuras 47 y 48, significa que no existe ese lenguaje cargado.

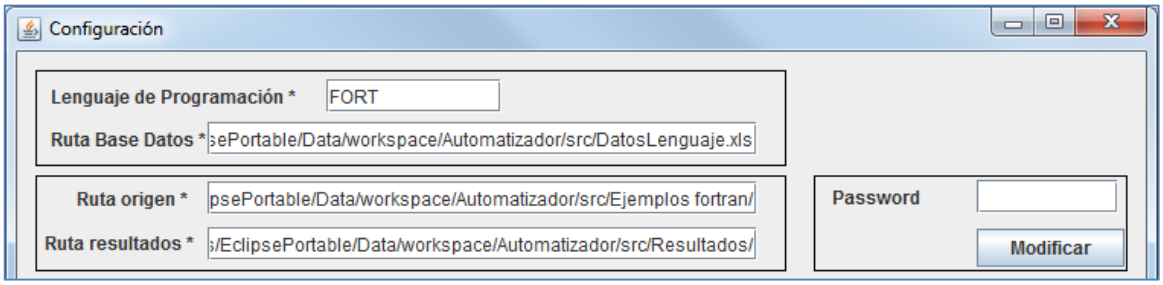


Figura 47. Lenguaje no existente en ninguna hoja de la ruta Excel

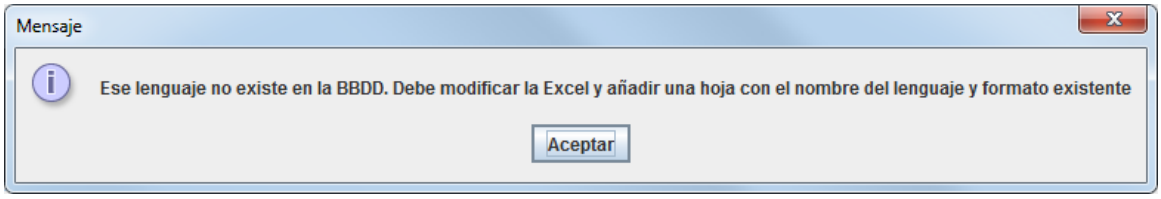


Figura 48. Mensaje de error de lenguaje

Si queremos añadir un nuevo lenguaje, se deberá nombrar en una nueva hoja en la Excel de datos, y añadir los datos pertinentes en una serie de columnas, ver Figura 49 y 50.

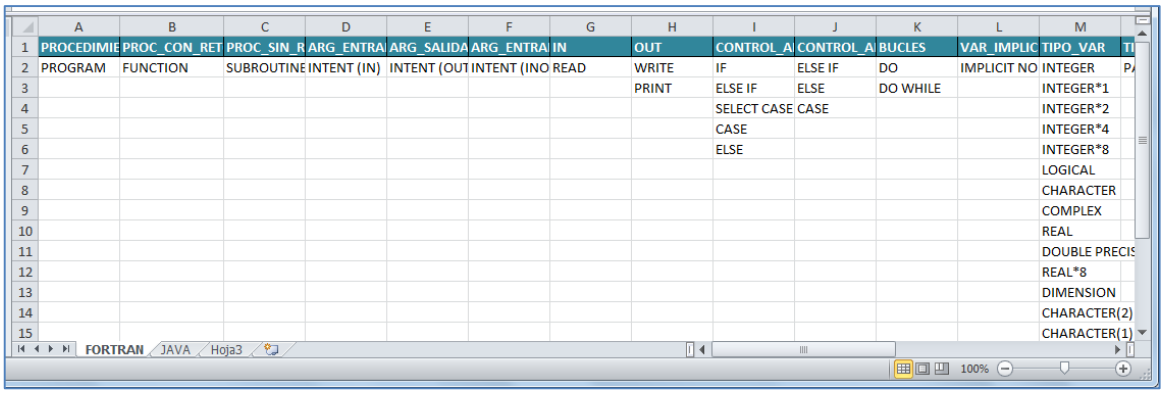


Figura 49. Excel de datos, lenguaje FORTRAN

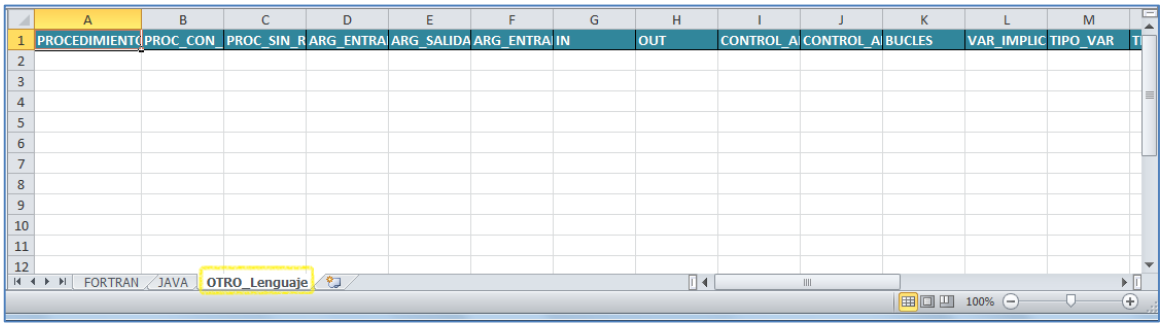


Figura 50. Creación de nuevo lenguaje en la BBDD

Los contenidos a introducir en cada columna se describirán en el Anexo, ver Manual de usuario.

4.6. Resultados

Esta herramienta es aplicable a cualquier asignatura de programación incluso si es susceptible de sufrir una modificación en el lenguaje de programación con el que se imparte. Pudiendo soportar seudocódigo. (Requisito RF-03).

Además, se pueden desactivar ciertos aspectos o criterios a voluntad según las necesidades pedagógicas propias del progreso en la asignatura.

Los resultados de la evaluación (según los pesos aplicados) aparecen en un pop up al finalizar la ejecución de la herramienta y, a su vez, se almacenarán en la ruta que el usuario especificó en la interfaz:

Figura 51. Ventana con la puntuación resultante de la evaluación

Se creará un archivo de tipo texto por cada fichero que contenga la ruta origen introducida, además de otro archivo con la documentación automática, ver Figura 52.

Podemos acceder a los resultados y visualizar los errores de código resultantes ver Figuras 53, 54 y 55. Al final del documente se añade la misma información de la Figura 51.

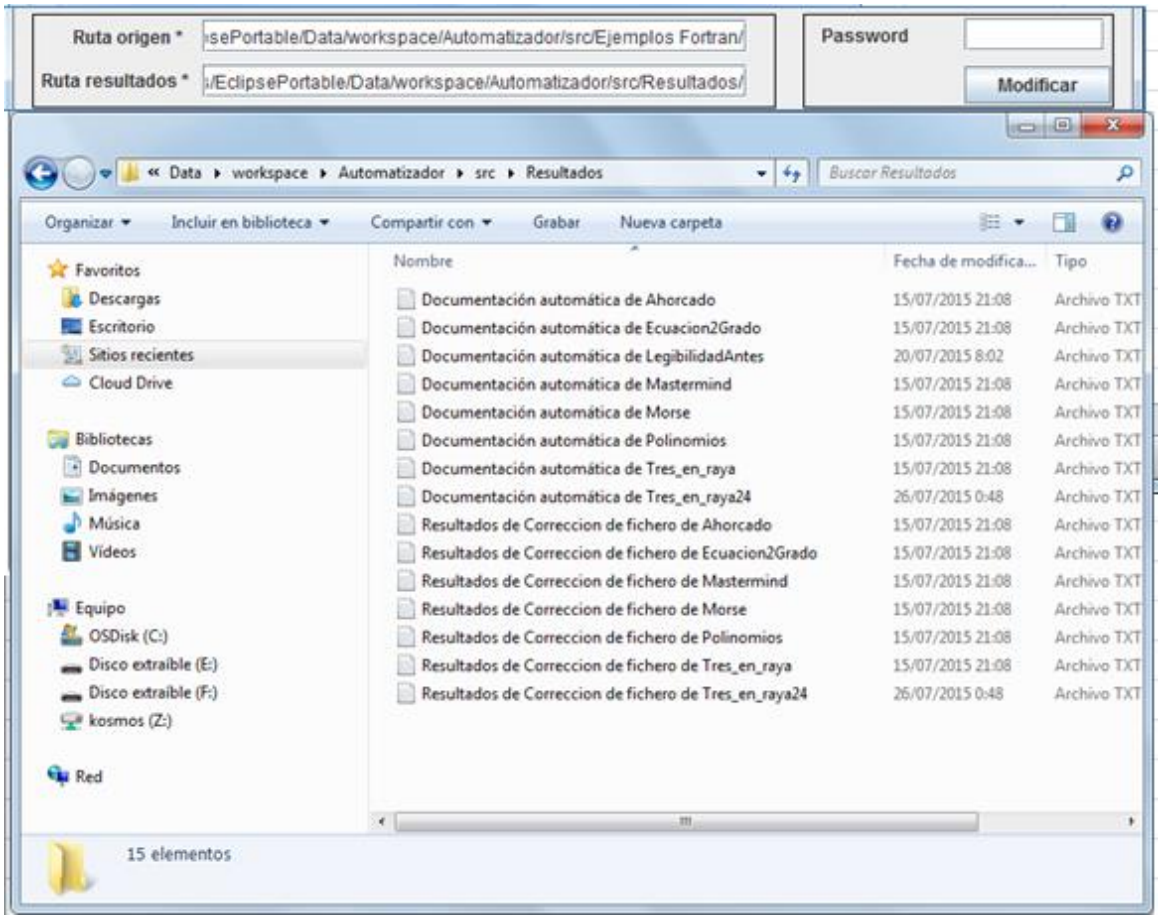
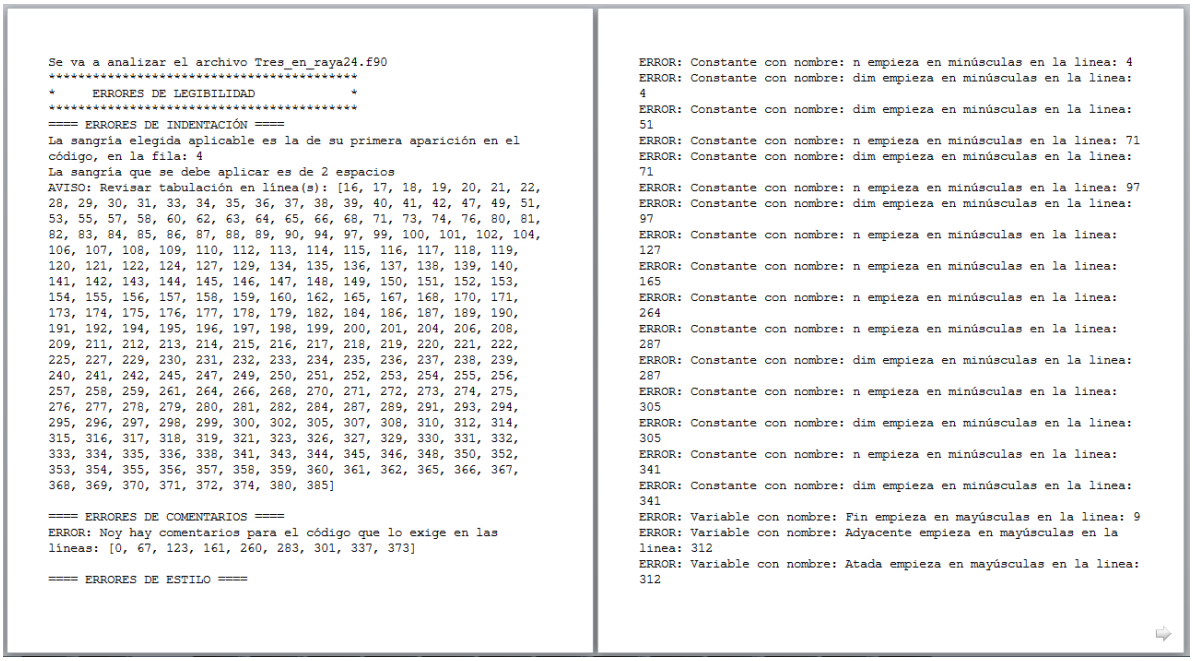


Figura 52. Resultados creados en la ruta especificada



```
Se va a analizar el archivo Tres_en_raya24.f90
*****************************************
*     ERRORES DE LEGIBILIDAD            *
*****************************************
==== ERRORES DE INDENTACIÓN ====
La sangría elegida aplicable es la de su primera aparición en el
código, en la fila: 4
La sangría que se debe aplicar es de 2 espacios
AVISO: Revisar tabulación en línea(s): [16, 17, 18, 19, 20, 21, 22,
28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 47, 49, 51,
53, 55, 57, 58, 60, 62, 63, 64, 65, 66, 68, 71, 73, 74, 76, 80, 81,
82, 83, 84, 85, 86, 87, 88, 89, 90, 94, 97, 99, 100, 101, 102, 104,
106, 107, 108, 109, 110, 112, 113, 114, 115, 116, 117, 118, 119,
120, 121, 122, 124, 127, 129, 134, 135, 136, 137, 138, 139, 140,
141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153,
154, 155, 156, 157, 158, 159, 160, 162, 165, 167, 168, 170, 171,
173, 174, 175, 176, 177, 178, 179, 182, 184, 186, 187, 189, 190,
191, 192, 194, 195, 196, 197, 198, 199, 200, 201, 204, 206, 208,
209, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222,
225, 227, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
240, 241, 242, 245, 247, 249, 250, 251, 252, 253, 254, 255, 256,
257, 258, 259, 261, 264, 266, 268, 270, 271, 272, 273, 274, 275,
276, 277, 278, 279, 280, 281, 282, 284, 287, 289, 291, 293, 294,
295, 296, 297, 298, 299, 300, 302, 305, 307, 308, 310, 312, 314,
315, 316, 317, 318, 319, 321, 323, 326, 327, 329, 330, 331, 332,
333, 334, 335, 336, 338, 341, 343, 344, 345, 346, 348, 350, 352,
353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 365, 366, 367,
368, 369, 370, 371, 372, 374, 380, 385]

==== ERRORES DE COMENTARIOS ====
ERROR: Noy hay comentarios para el código que lo exige en las
líneas: [0, 67, 123, 161, 260, 283, 301, 337, 373]

==== ERRORES DE ESTILO ====
```

```
ERROR: Constante con nombre: n empieza en minúsculas en la linea: 4
ERROR: Constante con nombre: dim empieza en minúsculas en la linea:
4
ERROR: Constante con nombre: dim empieza en minúsculas en la linea:
51
ERROR: Constante con nombre: n empieza en minúsculas en la linea: 71
ERROR: Constante con nombre: dim empieza en minúsculas en la linea:
71
ERROR: Constante con nombre: n empieza en minúsculas en la linea: 97
ERROR: Constante con nombre: dim empieza en minúsculas en la linea:
97
ERROR: Constante con nombre: n empieza en minúsculas en la linea:
127
ERROR: Constante con nombre: n empieza en minúsculas en la linea:
165
ERROR: Constante con nombre: n empieza en minúsculas en la linea:
264
ERROR: Constante con nombre: n empieza en minúsculas en la linea:
287
ERROR: Constante con nombre: dim empieza en minúsculas en la linea:
287
ERROR: Constante con nombre: n empieza en minúsculas en la linea:
305
ERROR: Constante con nombre: dim empieza en minúsculas en la linea:
305
ERROR: Constante con nombre: n empieza en minúsculas en la linea:
341
ERROR: Constante con nombre: dim empieza en minúsculas en la linea:
341
ERROR: Variable con nombre: Fin empieza en mayúsculas en la linea: 9
ERROR: Variable con nombre: Adyacente empieza en mayúsculas en la
linea: 312
ERROR: Variable con nombre: Atada empieza en mayúsculas en la linea:
312
```

Figura 53. Ejemplo de Resultados de la herramienta para un fichero con código FORTRAN (I)

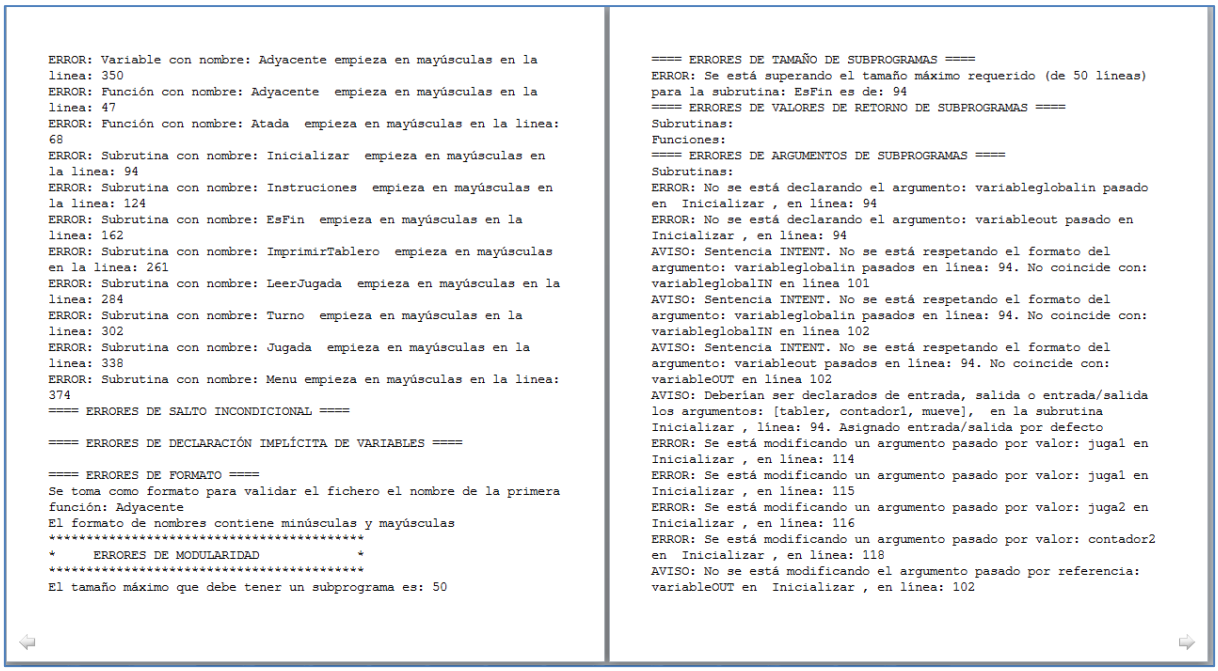


Figura 54. Ejemplo de Resultados de la herramienta para un fichero con código FORTRAN (II)

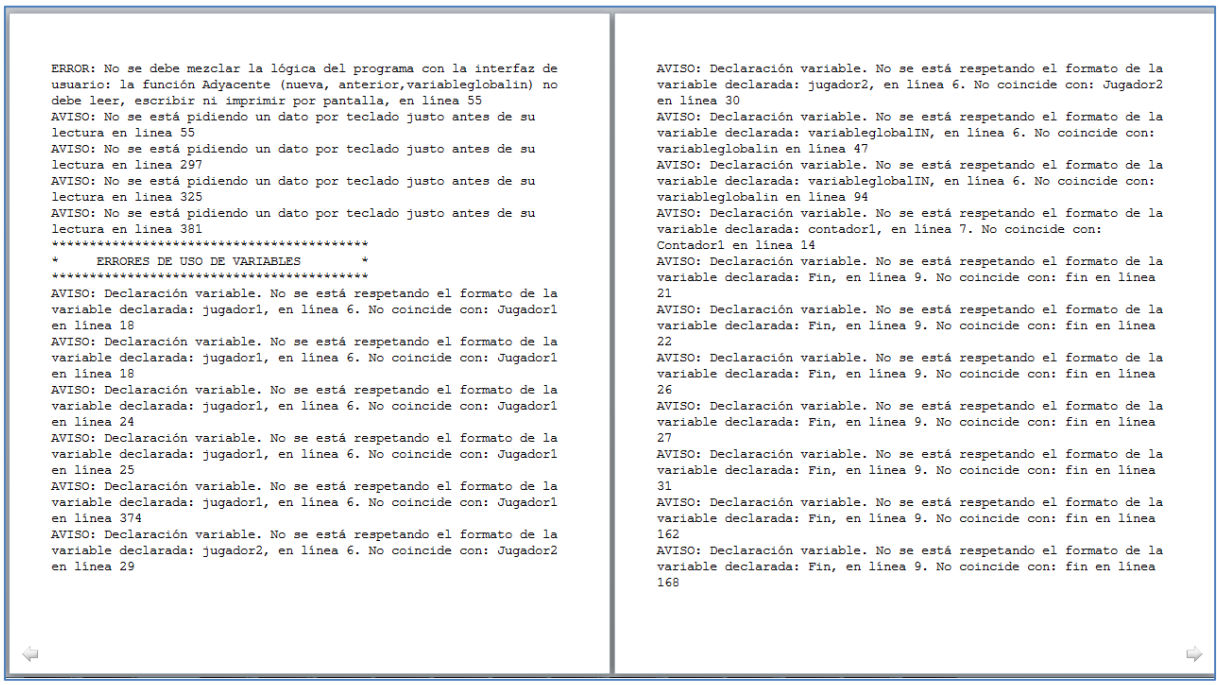


Figura 55. Ejemplo de Resultados de la herramienta para un fichero con código FORTRAN (III)

Unos mensajes irán precedidos de la palabra ERROR, por considerarse más críticos, que los AVISOS, que pueden recoger temas de estilo o que sirven al alumno como apoyo a la hora de poner foco en un posible error, aunque puede que no sea un error en sí (como en el caso de la mala práctica de nombrar una variable local igual que una global).

4.7. Documentación automática

La herramienta genera de documentación automática a partir de comentarios específicos embebidos en el código del alumno (Requisito RF-02).

Se completan las funcionalidades de la herramienta para que pueda proporcionar información, de forma automatizada, sobre la corrección de un programa a partir de juegos de prueba. Estos serán públicos con el propósito de que los alumnos puedan validar los resultados de sus prácticas, y privados para las revisiones por parte de los profesores, ver Figura 55.

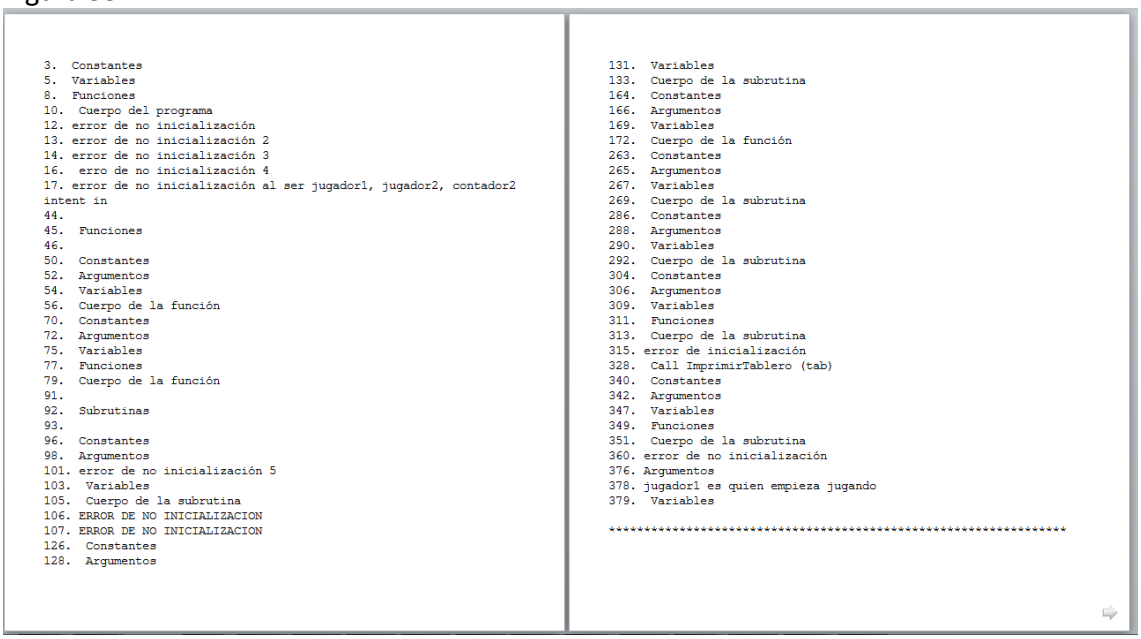


Figura 56. Ejemplo de Documentación automática

4.8. Decisiones de diseño

Para la conseguir el objetivo de configurabilidad en la herramienta, esto es, para que se pueda adaptar a todos los lenguajes de programación de las diferentes asignaturas, se han incluido las siguientes premisas, que además aportarán buenas prácticas al alumno, a la hora de la escritura de código:

Para el lenguaje JAVA:

- Para indicar el final de una clase, método, sentencias de control,… se incluirá un comentario indicando:

  }// nombre_clase
  }//nombre_método
  }//for
  }//if

  Y seguido de los comentarios que se deseen a continuación. Es una buena práctica también para la trazabilidad de sentencias de control, bucles, que exigen anidación en la indentación, y además soluciona el problema de identificar el final de los mismos. Por ejemplo:

```
                            }else{//la línea es la 0 no continuo hacia arriba en la búsqueda de comentarios
                                lineasAComentar.addElement(i);
                                añadirErrorComentarios();
                                break;
                            }//else
                        }//if la linea tiene comentarios
                    }//for que recorre los identificadores con los que comienzan los comentarios
                }//if estoy dentro del procedimiento
            }// if la línea contiene el identificador de procedimiento
        }//for con todos los identificadores de procedimientos
    }//for recorre el texto a corregir
    if(!lineasAComentar.isEmpty()){
        String mensajeErrorComentarios="ERROR: Noy hay comentarios para el código que lo exige en las líneas: "+linea
        añadirTextoLegibilidad(mensajeErrorComentarios);
        //añadirErrorComentarios();
    }//if el vector no está vacío
    return lineasAComentar;
}//validarComentariosAntes
```

Figura 57. Ejemplo de fin de método y sentencias de control alternativo y repetitivo en JAVA

Para la base de datos:

Se empezó a diseñar el acceso a una base de datos en SQL, y posteriormente mediante Access, pero dada la simplicidad de los datos, y viendo que no exigía datos relacionales se decidió que era más fácil, tanto para el usuario de la herramienta como para el diseño, hacerlo a través de una Excel de datos.

El documento Excel es manejable, se puede modificar fácilmente y el alumno puede consultar en todo momento qué instrucciones hay en cada columna pues es una herramienta habitual para el alumnado. Esta Excel se cargará en la ruta especificada junto con los propios ficheros del código del alumno.

Finalmente tenemos el siguiente diagrama de clases, donde se ven claramente diferenciadas las distintas capas que cumplen la arquitectura MVC. Sólo se han añadido los métodos, suprimiendo los atributos de clase, ver Figura 58.

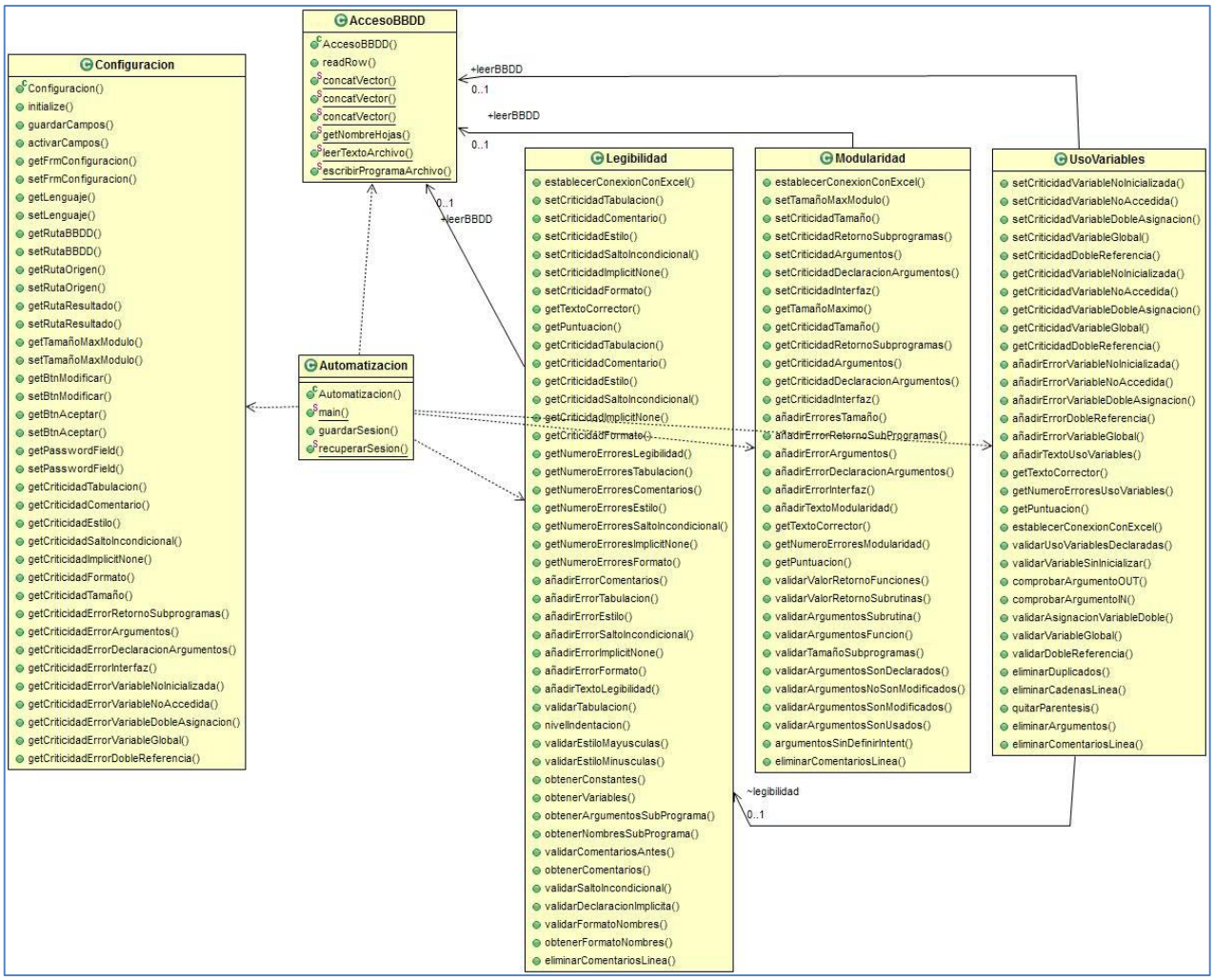


Figura 58. Diagrama de clases

5. VALIDACIÓN Y RESULTADOS

5.1. Estrategia de pruebas

5.1.1. Pruebas unitarias

Gracias al modelo MVC se facilita la labor de realización de pruebas unitarias de cada componente individual, y más aún de cada requisito particular, por lo que esta arquitectura permite aplicar desarrollo guiado por pruebas.

Aplicando las buenas prácticas de programación expuestas en capítulos anteriores, ya sea para el diseño de la herramienta o para el aprendizaje de la programación, se ha perseguido la calidad del software siguiendo el diagrama del proceso de validación durante el diseño de la herramienta, ver Figura 59:

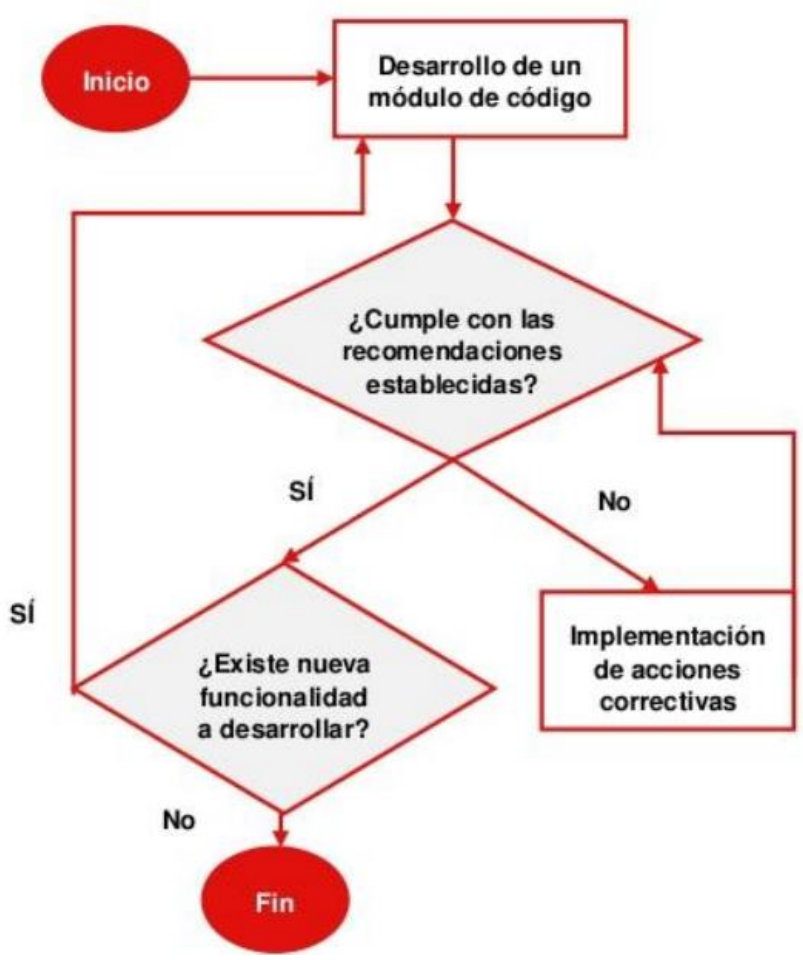


Figura 59. Diagrama del proceso de desarrollo y calidad del SW [MIE]

5.1.2. Pruebas de integración

Una vez se finalizan las pruebas unitarias, se procede con las pruebas para comprobar que el conjunto de módulos funciona conjuntamente.

En esta fase, se añadió la BBDD en formato Excel con las instrucciones a cargar para lenguaje JAVA.

Se tuvo que modificar el diseño del código para adaptar las instrucciones de Fortran a la lectura de columnas en la Excel de datos, también se ajustó el modelo fiel al patrón MVC, se mejoró el rendimiento realizando las llamadas a BBDD desde la capa de Modelo.

5.1.3. Pruebas de validación

Finalmente las pruebas de validación comprueban el correcto funcionamiento del conjunto total de requisitos y casos de uso que permiten certificar la herramienta.

Se puede concluir tras las pruebas realizadas con una pequeña muestra que sirve de ejemplo de programas de alumnos que el mayor número de errores se detectan en el módulo de Legibilidad, seguido del de UsoVariables y por último el de Modularidad. Así como también, se ha pasado el código de esta herramienta la propia evaluación del código de la herramienta pasada por la herramienta para su corrección.

5.2. Resultados

Tras las pruebas de validación sobre código de alumnos en lenguaje de programación Fortran, se analizan los resultados obtenidos de la siguiente manera:

	Errores		
Programas	Legibilidad	Modularidad	UsoVariables
Ahorcado	57	4	0
Mastermind	157	16	53
Polinomios	242	20	143
Tres_en_raya	187	25	42
Ecuacion2Gra	35	0	17
Morse	85	1	43
Totales	**763**	**66**	**298**

Tabla 48. Resultados de la herramienta sobre código de alumnos en Fortran

Se observa un mayor número de errores de Legibilidad, esto puede ser debido a que, por ejemplo, un error de indentación se arrastra durante todo el código.

A continuación le siguen los errores de uso de variables, donde se observa que la fuente principal y motivo de errores es el acceso a variables globales, resultado de nombrar de igual forma las variables locales de las globales.

Representamos gráficamente estos resultados:

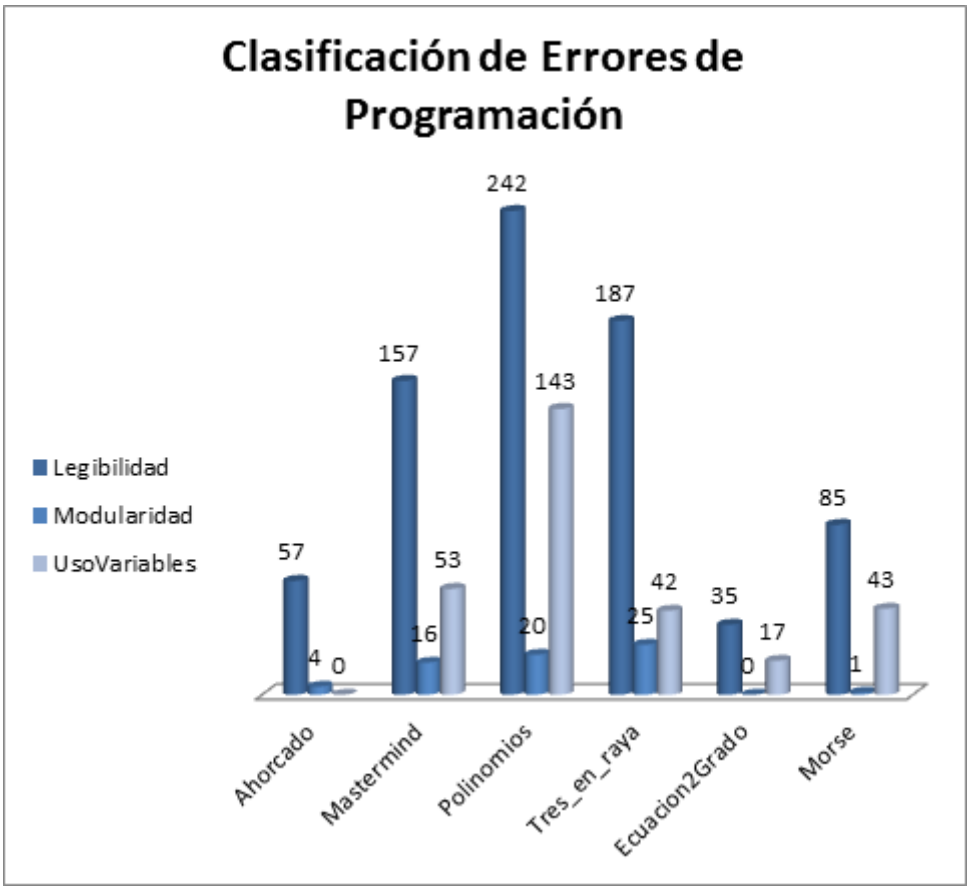


Figura 60. Clasificación de los errores de programación

6. PLANIFICACIÓN Y PRESUPUESTO

A continuación, se presenta la planificación inicial y final del proyecto utilizando diagramas Gantt. Se incluye un presupuesto con la estimación de los costes de realización de esta herramienta.

6.1. Planificación Estimada

Se fasea la gestión del proyecto en las siguientes etapas:

- Inicio:
  - Planteamiento inicial: Visitas al tutor y planteamiento de la herramienta, toma de contacto con el nuevo lenguaje (Fortran).
  - Resolución de dudas: con respecto al planteamiento inicial, intercambio de opiniones sobre el estado del arte que enmarca el proyecto, soluciones actuales y recepción de documentación de la asignatura.
  - Decisiones de implementación: Se comienza la estructura de clases intentando adaptarlo al modelo de arquitectura decidido.
- Programación Fase I:
  - Implementación de los requisitos de cada módulo lógico (Legibilidad, Modularidad y Uso de Variables).
  - Pruebas unitarias a la finalización de cada requisito de cada módulo aplicando ficheros reales de códigos de alumnos.
  - Corrección de errores: consulta de dudas, agrupando varias y solicitando resolución en tutorías presenciales.
- Punto de Control tutor: resolución de un conjunto que pueden ser triviales o bloqueantes para el diseño.
- Programación Fase II:
  - Decisión del diseño de la BBDD: barajar las opciones para la creación de una base de datos y tomar la solución óptima.
  - Implementación de la BBDD en base a lo anterior
  - Adaptar el código para integrar la BBDD y realizar los cambios necesarios en previsión de cualquier otro lenguaje (JAVA en principio)
  - Pruebas de integración de todos los módulos con acceso a la BBDD
  - Corrección de errores y ajustes del diseño para mejorar el rendimiento.
- Punto de Control con el Tutor: presentación de las mejoras introducidas y planteamiento de problemas encontrados
- Documentación Fase I: Se plantea la estructura de la memoria
  - Estado del Arte: Redacción en base a los conocimientos aplicados para el desarrollo de la herramienta, incorporando nuevos conceptos adquiridos.
  - Documentar el manejo y funcionamiento de la herramienta diseñada
- Punto de Control con el Tutor: revisión de los puntos a mejorar
- Documentación Fase II:
  - Estado del Arte con el feed-back del tutor
  - Especificaciones de la herramienta con el feed-back del tutor

Teniendo en cuenta la que la realización del proyecto se ha tenido que compaginar con la realización de actividades laborales a tiempo completo, se ha realizado una media de 22 días útiles al mes empleando 4 horas diarias de trabajo.

A continuación el diagrama de Gantt (Figura 61) con la previsión de horas estimadas para la realización del proyecto, estimando para la realización del proyecto con un total de 169 días de trabajo (8 meses).

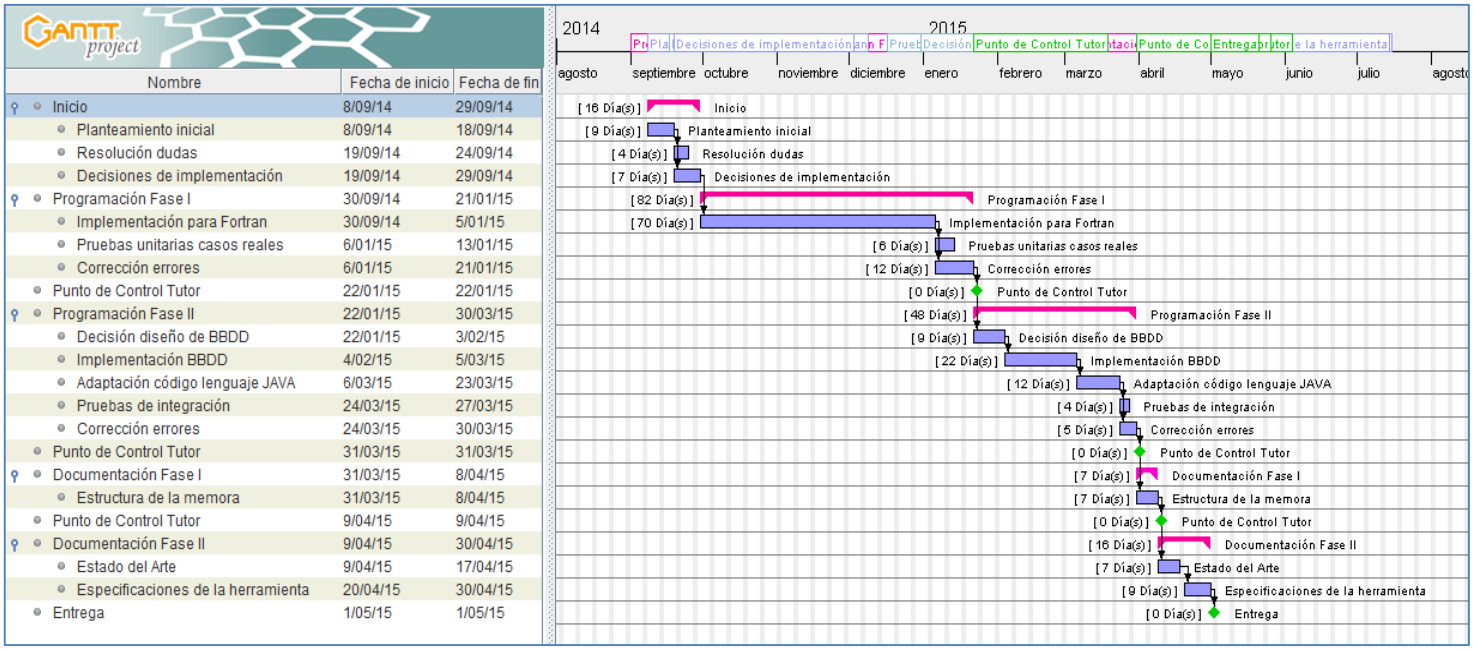


Figura 61. Planificación GANTT estimada del proyecto

6.2. Planificación Real

En la práctica, no se cumplió la anterior estimación resultando un retraso de 3 meses debido a otros compromisos e imprevistos. Esto afectó a la fase I de programación. Sin embargo, se intentó cumplir con los objetivos para no demorar demasiado la entrega reduciendo los tiempos en la fase de documentación. Resultando un total de 227 horas trabajadas, ver Figura 62.

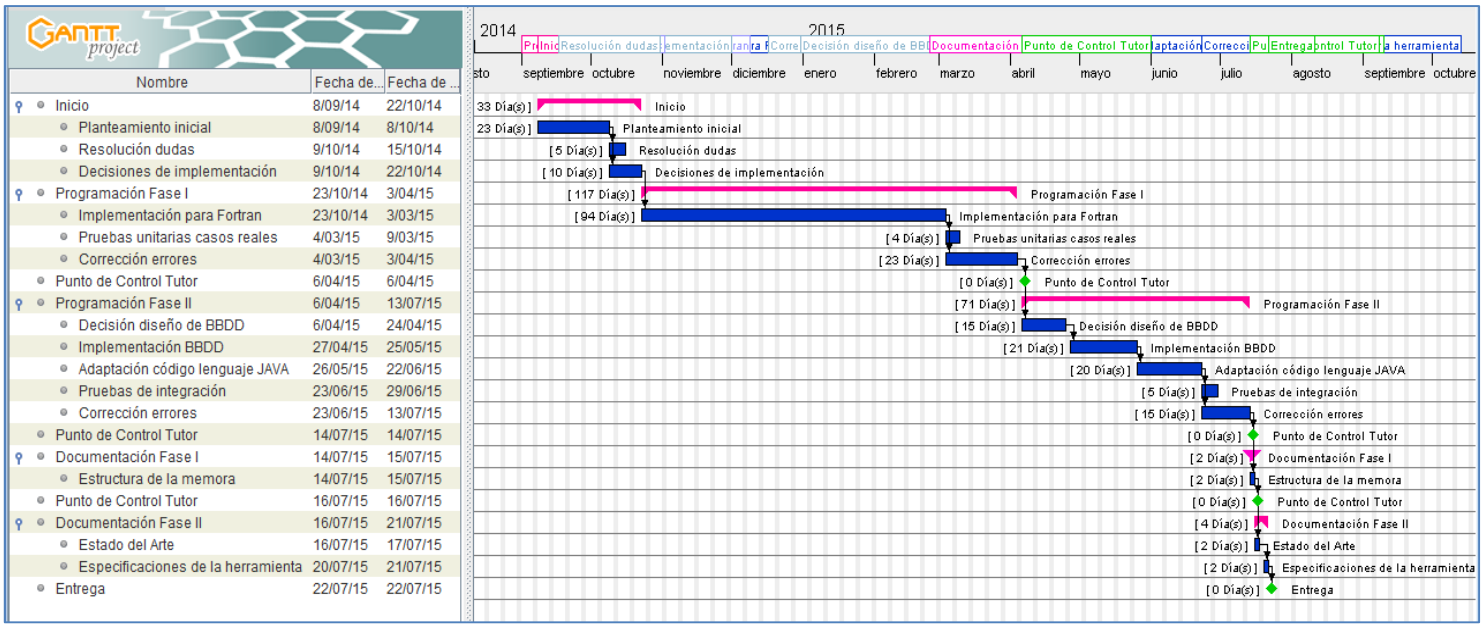


Figura 62. Planificación GANTT real del proyecto

En el siguiente Apartado se hace una estimación de los costes asociados a la realización del proyecto mediante un Presupuesto detallado.

6.3. Presupuesto

UNIVERSIDAD CARLOS III
Escuela Politécnica Superior
UNIVERSIDAD CARLOS III
LEGANÉS
Fecha 30/07/15

TÍTULO DEL PROYECTO: "Aprendizaje dirigido por detección automática por errores de programación"

MATERIAL TÉCNICO	Cantidad	Precio/ud.	Importe
Ordenador Portátil SONY VAIO PRO 13 LAPTOP SVP132A1CW	1	€ 1,899,90	€ 368,00*
Disco almacenamiento y Back Up	2	€ 90,00	€ 180,00
Pen Drive	2	€ 9,00	€ 18,00
SOFTWARE			
Eclipse Indigo	1	€ 0,00	€ 0,00
Eclipse Luna	1	€ 0,00	€ 0,00
Microsoft Office Home and Student	1	€ 79,99	€ 79,99
Adobe Acrobat Pro (precio al mes)	2	€ 18,14	€ 36,28
Photoshop (precio al mes)	2	€ 24,19	€ 48,38
GASTOS de Producción desarrollo PFC			
PACK HOGAR (ADSL, Móvil 4G, LUZ) días	300	€ 2/día	€ 600,00
Traslados y kilometraje Madrid-Leganés Leganés-Madrid 50 KM a 0,21€/KM	10	€ 10,50	€ 100,50
Alquiler mensual espacio de trabajo	10	€ 300,00	€ 3.000,00
HONORARIOS			
Honorarios en horas. Ingeniera Técnica de Telecomunicación. Especialidad Sistemas de Telecomunicaciones.	904	€ 19,00	€ 17.176,00

MATERIAL TÉCNICO	Cantidad	Precio/ud.	Importe
		Subtotal	€ 21.607,15
		Total	**€21.607,15**

* Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo

D = % del uso que se dedica al proyecto (en este caso 100%)

Todos los importes incluyen IVA.

El presupuesto total de este proyecto asciende a la cantidad de 21.607,15 EUROS.

Leganés a 22 de Julio de 20015

Atentamente, la ingeniera proyectista

Fdo. Ana González Ibáñez.

7. CONCLUSIONES Y TRABAJOS FUTUROS

La realización de este proyecto cubre los objetivos planteados al comienzo del mismo, en donde se pretendía crear una herramienta que sirva de aprendizaje dirigido al alumno en la detección de errores de programación, así como también, sirva al profesor en la labor de evaluación de los programas del alumnado.

El empleo de esta herramienta facilita la unificación de criterios aplicados por los profesores de los diferentes grupos. Establece una métrica en la criticidad de los errores y por tanto da idea al alumno de dónde debe poner más foco. Esta información es relevante tanto para la fase de evaluación del alumnado, como retroalimentación que permita conseguir el máximo aprovechamiento en las siguientes sesiones docentes, ya que al detectar los puntos especialmente conflictivos puede insistir en ellos, y profundizar donde resulte más conveniente para el estado actual de evolución de la clase.

Adicionalmente, se reduce el tiempo que el profesor dedica a corregir los fallos, cuyo asesoramiento ha sido ahora automatizado y puede dedicarse más profundamente a atender otros problemas más específicos durante las tutorías que no han sido contemplados y que son de difícil o imposible automatización

Si bien es cierto que existen herramientas en la detección de errores, ya sea el propio compilador o un entorno de desarrollo más o menos sofisticado, esta nueva herramienta introduce un elemento innovador en la enseñanza de la programación, fijando las normas de estilo en el alumno, aumentando la autonomía y el rendimiento pues el alumno refuerza los conocimientos adquiridos en clase utilizando en cualquier lugar y momento un “asesor” que sirve de guía en períodos no lectivos para aquellos errores indetectables en la ejecución y que determinan las buenas prácticas de un futuro desarrollador.

La herramienta ofrece un entorno amigable y motivador, mejorando la autonomía, el rendimiento del alumno y buscando la reducción de errores mediante la justificación del problema y proporcionando sugerencias para una programación correcta o simplemente alternativa.

No cabe duda de que, como alumna, durante el desarrollo este proyecto tanto a nivel de programación como de aprendizaje, la herramienta ya ha comenzado su fase de pruebas de aceptación de usuario. El ejercicio del diseño de la propia herramienta ha planteado dudas de concepto hasta ahora no planteadas, de modo que la herramienta da una vuelta al concepto educativo, ofreciendo un servicio proactivo, rehaciendo en muchas ocasiones la implementación debido a la conciencia de los errores cometidos.

Los resultados de la herramienta han sido satisfactorios, pues se consigue una aplicación funcional y portable que puede ser utilizada en cualquier curso de programación dada su alta configurabilidad y mantenibilidad.

7.1. Líneas futuras de investigación

Como ya se ha planteado en el corpus de la memoria las línea de trabajo futuras que surgen tras el estudio de los errores de programación son innumerables, pero daremos aquí unas pinceladas de las posibles vías de mejora y ampliación de esta herramienta.

- **Para aumentar el alcance extensivo de la herramienta**: Dado que este asesor puede ser aplicado a cualquier entorno docente que comparta los mismos objetivos que las asignaturas piloto, se podría utilizar en otras asignaturas, como Programación de Ingeniería en Informática, Programación de Ingeniería Industrial, y en otras titulaciones con independencia del lenguaje, dado que las reglas de estilo y los fines perseguidos son comunes a todas ellas.

- **Para aumentar el alcance en profundidad de la herramienta**: Se pueden añadir en la herramienta la revisión de los siguientes de errores recurrentes adicionales para mejorar la calidad. Según el Ministerio de Industria, Energía y Turismo, las buenas prácticas para la calidad de la programación se consigue reduciendo [MIE]:

    - **Código duplicado.** La existencia de código repetido en diferentes puntos de un programa implica mayores costes de mantenibilidad, duplicidad de errores y de código a probar
    - **Complejidad.** La complejidad es el número de posibles caminos lógicos que tiene cada componente, es decir, las sentencias selectivas (if, case), sentencias iterativas (while, for), lanzamiento y tratamiento de excepciones (throw, catch), etc. A mayor complejidad, un sistema es más difícil de modificar y de probar.
    - **Eficiencia.** Mejorando el uso de los recursos, abarca los siguientes comportamientos, entre otros muchos**:**
        - **Comportamiento temporal:** La capacidad del producto software para proporcionar tiempos de respuesta, tiempos de proceso y potencia apropiados, bajo condiciones determinadas.
        - Utilización de recursos: La capacidad del producto software para usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su función bajo condiciones determinadas.
        - Métodos que concatenan strings utilizando ‘+’ dentro de un bucle.
    - **Mantenibilidad.** facilidad que tiene el sistema para ser adaptada a un cambio en el entorno, así como su capacidad para corregir fallos y para probarlos. Para mejorar la mantenibilidad hay que evitar:
        - **Bloques vacíos.** Esta regla comprueba la existencia de bloques de tipo if, while o finally vacíos, es decir, en los que no se ejecuta ninguna acción. A la hora de desarrollar, se deberían omitir este

tipo de construcciones.

- **Bloques sentencias de control incondicionales.** Es decir, bloques que son siempre verdaderos o false.
- **Bloques que no cumplan con los criterios de fin de línea o de construcción de código.** Esto ayuda a encontrar errores con la herramienta y a clarificar el código haciéndolo más comprensible.
- **Longitud de línea.** Vigilar la longitud de las líneas de código que no superen los 80 caracteres.
- **Variables cortas.** Se recomienda evitar el uso de variables con un nombre muy corto intentando que aporte valor y ayudando así a la comprensión.
- **Convención de nombres de variables.** Añade a la herramienta que el formato de las variables no debe contener guiones bajos.

Para llevar esto a cabo hay que establecer una métrica y unos criterios de aceptación del Software para establecer el riesgo de cometer cada tipo de error.

- **Para mejorar la labor de aprendizaje de la herramienta:**
  - Se puede añadir una historia de trabajo individual por alumno que permita el seguimiento real y la evolución de cada alumno adaptando así el aprendizaje ad-hoc. De modo que se puedan utilizar distintos perfiles de usuario basados en errores de programación.
  - Se puede añadir un histórico de los errores más comunes en general dentro de una clase o curso y sirviendo la herramienta como medidor de aprendizaje, en caso de que el profesor quiera emplear distintos métodos de enseñanza.
  - Permite la realimentación de la información que maneja la herramienta con las nuevas aportaciones y conocimientos de los desarrolladores.
- **Para aumentar y promover el uso de la herramienta:**
  - Modificaciones en la interfaz que hagan de su uso un juego para el alumno, establecer un ranking de puntuaciones, incorporando gráficos animados, etc.
  - Añadiendo mensajes personalizados dado el historial del alumno.
  - El alumno también puede completar la Excel de datos como instrumento de aprendizaje del lenguaje de programación a estudiar.

8. REFERENCIAS Y BIBLIOGRAFÍA

8.1. Referencias

[AEN] AEN/CTN 71 - TECNOLOGÍA DE LA INFORMACIÓN
Disponible [Internet]:
<http://www.aenor.es/aenor/normas/ctn/fichactn.asp?codigonorm=AEN/CTN%2071#.VbSC3_msWO0> [16 de Julio de 2015]

[ACM] The Joint Task Forcé on Computing Curricula ACM/IEEE, Computer Society Association for Computing Machinery, 2001.
Disponible [Internet]: <http://www.acm.org/education/curric_vols/cc2001.pdf> [22 de Julio de 2015]

[ANG] El e-learning en el siglo XXI. Investigación y práctica. T. Anderson, D.R. Garrison. Octaedro. Barcelona, España.
Disponible [Internet] <www.octaedro.com/downloadf.asp?m=10057.pdf> [01 de septiembre de 2015]

[BOE] Ley Orgánica 6/2001, de 21 de diciembre de Universidades (BOE 24-12-2001) Disponible [Internet]: <http://www.boe.es/diario_boe/txt.php?id=BOE-A-2001-24515> [22 de Julio de 2015]

[BO2] Real Decreto 1538/2006. BOE-A-2007-92. Capítulo IV. Artículo 18. Apartados 2 y 4.
Disponible [Internet]: <http://www.um.es/docencia/barzana/MASTER-INFORMATICA-II/Metodos-y-tecnicas-didacticas-para-la-ensenanza-de-la-informatica.html> <http://www.boe.es/diario_boe/txt.php?id=BOE-A-2007-92> [22 de Julio de 2015]

[CCE+]Cerrada, J.A.; Collado, M; Estívaris, J.F. y Gómez, S. R.: Fundamentos de programación con Modula-2. Editorial Centro de Estudios Ramón Areces, S. A. Madrid, 2000.

[DET] Détienne, F. Software-design cognitive aspects. Springer 2002

[ERR] Errores en tiempo de Ejecución. 2012.
Disponible [Internet]: <https://sites.google.com/site/tecnologicodetuxtlagutierrez/3-6-errores-en-tiempo-de-ejecucion> [16 de julio de 2015]

[FPP+] Aprendizaje de la programación guiado por los errores de compilación. Actas de las XX JENUI. Oviedo, 9-11 de julio 2014 ISBN: 978-84-697-0774-6 Páginas: 371-378 Departamento de Informática. Departamento de Informática. Carlos Fernández Medina, Juan Ramón Pérez Pérez, Mª del Puerto Paule-Ruiz, Víctor Álvarez Garcia.
Disponible [Internet]: < http://www.aenui.net/jenui2014/78.pdf> [16 de Julio de 2015]

[FRE] La Arquitectura Modelo - Vista - Controlador como base de la plataforma de servicio. Miguel Ángel Frechoso, 2011.
Disponible [Internet]: <http://blog.cubenube.com/2011/11/la-arquitectura-modelo-vista.html> [16 de julio de 2015]
<e-archivo.uc3m.es>

[GAR] Entorno de Desarrollo Integrado (IDE), 2013.
Disponible [Internet]: <https://fergarciac.wordpress.com/> [16 de julio de 2015]

[IEE] IEEE Standards Collection Software Engineering. IEEE, 1997.

[ISI] Extracto del documento INTERNATIONAL STANDARD ISO/IEC 19796-1 First edition 2005-11-01.
Disponible [Internet]:
<file:///C:/Users/x069403/Downloads/EXT_TJWPVUCYUFWC5F5FI34J.pdf>

[ISO] Norma UNE-EN ISO/IEC 19796-1:2010.
Disponible [Internet]:
<http://www.aenor.es/aenor/normas/normas/fichanorma.asp?tipo=N&codigo=N0045773#.Va5dsPmsWW0> [16 de julio de 2015]

[LOR] Object-Oriented Software Metrics (Prentice Hall, 1994), M. Lorentz and J. Kidd

[MEL] Theory and Research-based Principles of Learning. Carnegie Mellon University.
Disponible [Internet]: <http://www.cmu.edu/teaching/principles/learning.html> [16 de julio de 2015]

[MIE] Buenas prácticas de calidad en el desarrollo de aplicaciones CALIDAD. Plataforma Avanza Local Soluciones AL efácil. Ministerio de Industria, Energía y Turismo.
Disponible [Internet]: <file:///C:/Users/x069403/Downloads/SCAU_TEC_CALIDAD.pdf>

[MOR] Entorno interactivo para el establecimiento de buenos hábitos y el entrenamiento personalizado de programadores noveles V. Moreno Pelayo, A. Granados Fontecha, A. Sanchis de Miguel. Departamento de Informática. Universidad Carlos III de Madrid. R. Peña Ros. Departamento de Ciencias de la Computación. Universidad de Alcalá

[NUE] Nuevas tecnologías para el aprendizaje . Estado del arte Rubén Edel Navarro. Capítulo 2 del libro. Editorial Pearson. pp.15-28. ISBN: 978-970-26-1213-1.
Disponible [Internet]:
<http://www.academia.edu/12285611/Las_nuevas_tecnolog%C3%ADas_para_el_aprendizaje_Estado_del_Arte> [01 de septiembre de 2015]

[PED] Pedagogía. Didáctica. Calidad del aprendizaje. Educación individualizada y socializada. Madurez. Motivación. Formación docente. Técnicas y Métodos de Enseñanza. Principios didácticos. Clasificación. Metodologías.
Disponible [Internet]:

<http://es.slideshare.net/adalbertomartinez/mtodos-y-tcnicas-de-enseanza-20561298> [16 de julio de 2015]
<http://html.rincondelvago.com/tecnicas-y-metodos-de-ensenanza.html> [25 de julio de 2015]

[PER] CLASIFICACION DE USUARIOS BASADA EN LA DETECCION DE ERRORES USANDO TECNICAS DE PROCESADORES DE LENGUAJE. Tesis doctoral. Universidad de Oviedo. Juan Ramón Pérez Pérez. 2006.
Disponible [Internet]:
<https://www.educacion.gob.es/teseo/mostrarRef.do?ref=394662> [01 de septiembre de 2015]
<http://www.powershow.com/view/28bf89-NzRjM/Clasificacin_de_usuarios_basada_en_la_deteccin_de_errores_usando_tcnicas_de_procesadores_de_lenguaje_powerpoint_ppt_presentation> [01 de septiembre de 2015]

[RAN] Tipos de errores en programación. De compilación o ejecución. Gestionados y no gestionados. Mario R. Rancel. Curso Bases de la programación Nivel II. 2006.
Disponible [Internet]:
<http://www.aprenderaprogramar.com/index.php?option=com_attachments&task=download&id=292> [16 de julio de 2015]

[REB] Aprendizaje autodirigido Propuesta de trabajo para la asignatura Usuario de WWW Miguel Rebollo Febrero, 2002.
Disponible [Internet]:
<http://users.dsic.upv.es/asignaturas/fade/oade/download/Aprendizaje%20autodirigido.pdf> [16 de julio de 2015]

[ROS] Principios de la Enseñanza y el aprendizaje. Universidad del Rosario. Colombia.
Disponible [Internet]:
<http://www.urosario.edu.co/cea/Profesores/Principios/#.VbOb9PmsWO0> [16 de julio de 2015]

[SAL] Métodos de Investigación (3ª. Edición) Salkid, N. (1998) Editorial Prentice Hall.
Disponible [Internet]: <http://www.monografias.com/trabajos15/metodos-ensenanza/metodos-ensenanza.shtml> [16 de julio de 2015]

[SCH] Tom Schorsch "CAP: an automated self-assessment tool to check Pascal programs for syntax, logic and style errors". March 1995
ACM SIGCSE Bulletin , Volume 27 Issue 1, 168-172

[TEC] CABERO, J. (2001): Tecnología educativa. Diseño y producción de medios, Barcelona, Paidós.
Disponible [Internet]:
<http://www.lmi.ub.es/te/any2004/documentacion/3_cabero.pdf> [16 de julio de 2015]

[UCE] El gran desafío gerencial – cómo controlar y reducir los gastos en informática y sistemas. Resumen ejecutivo.
Disponible [Internet]: <http://www.ucema.edu.ar/conferencias/download/CEMA-_Desafio_gerencial.pdf>

[VEG] Casos de uso UML. Miguel Vega. Universidad de Granada, Octubre 2010.
Disponible [Internet]: <http://es.slideshare.net/jpincay/diagramas-de-caso-de-uso-ejemplos>

8.2. Bibliografía

Finishers. Ironman Stories In Lanzarote. Silvia Domínguez Vidal, 2015. http://silviadominguez.com/finisherstheproject/

Wikipedia (2012), Consultado en: http: //es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado, el 21 de Enero del 2013.

Anderson, J. R., Conrad, F. G., Corbett, A. T. (1989). Skill acquisition and the LISP tutor. Cognitive Science, 13(4), 467-505.

S. N. Freund , Eric S. Roberts, Thetis: an ANSI C programming environment designed for introductory use, ACM SIGCSE Bulletin, v.28 n.l, p.300-304, March 1996

Cerrada, J.A.; Collado, M; Estívaris, J.F. y Gómez, S. R.: Fundamentos de programación con Modula-2. Editorial Centro de Estudios Ramón Areces, S. A. Madrid, 2000.

9. ANEXO

9.1. Glosario de términos

API	*Interfaz de Programación de Aplicaciones*
IDE	*Entorno de Desarrollo Integrado*
GCS	Gestión de la Calidad del Software
GUI	*Guía de Interfaz de Usuario*
MIE	*Modelos de Informática Educativa*
MVC	*Modelo Vista Controlador*
SQA	Software Quality Assurance
UAT	User Acceptance Testing

9.2. Manual de usuario

En este apartado se describe el Manual de Usuario de la herramienta de "Aprendizaje Dirigido por Detección Automática de Errores" que sirve de guía de uso de la herramienta para cualquier usuario.

9.2.1. Instalación de la herramienta en el equipo de usuario

La herramienta está diseñada en Java, por lo que hay dos opciones:

1. El docente puede distribuir el código a los alumnos, que deberá disponer de las clases siguientes:

   - Automatizacion.java (Controlador)
   - Configuracion.java (Vista)
   - Legibilidad.java (Modelo)
   - Modularidad.java (Modelo)
   - UsoVariables.java (Modelo
   - AccesoBBDD (BBDD)

   Se debe emplear un entorno de desarrollo compatible con Java: por ejemplo, Eclipse Juno (se recomienda una versión superior a la 3.5) donde deberá importarse las librerías POI de Apache para manipular ficheros Excel y WindowsBuilder para la interfaz gráfica.

   El profesor, también podrá facilitar la librería requerida que debe instalarse:

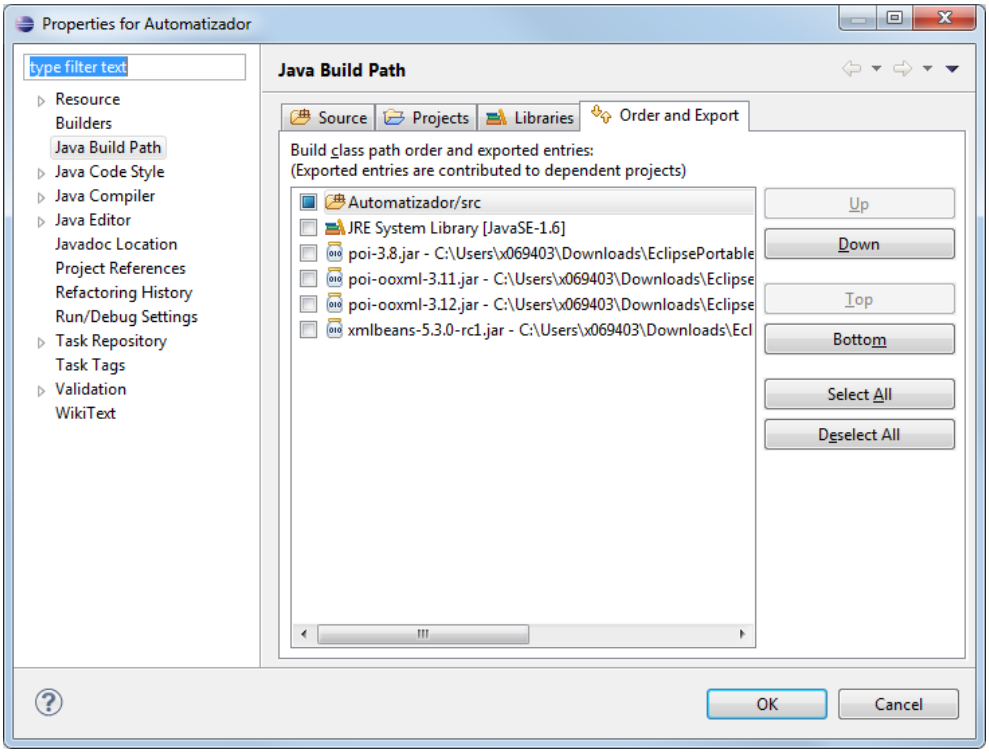


Figura 63. Configuración de Eclipse para uso de la herramienta (I)

A continuación se crea un nuevo proyecto Java y se le da el nombre que se desee, por ejemplo “Automatizador.java” y se pegan las clases en la ruta por defecto, normalmente dentro de: C:\...\Eclipse\Data\workspace\Automatizador\src\

2. El profesor puede crear un ejecutable y distribuirlo a los alumnos para mejorar la portabilidad de la herramienta

El profesor facilitará también el fichero Excel de la base de datos de la herramienta con los lenguajes precargados que el alumno guardará en su equipo local.

Se debe tener acceso de administrador en el equipo pues la herramienta crea un fichero temporal para guardar la sesión en C:/temp/sesión.txt

9.2.2. Ejecución de la herramienta

Una vez que tengamos instalada la herramienta siguiendo el apartado anterior, se seguirá el flujo normal de la herramienta:

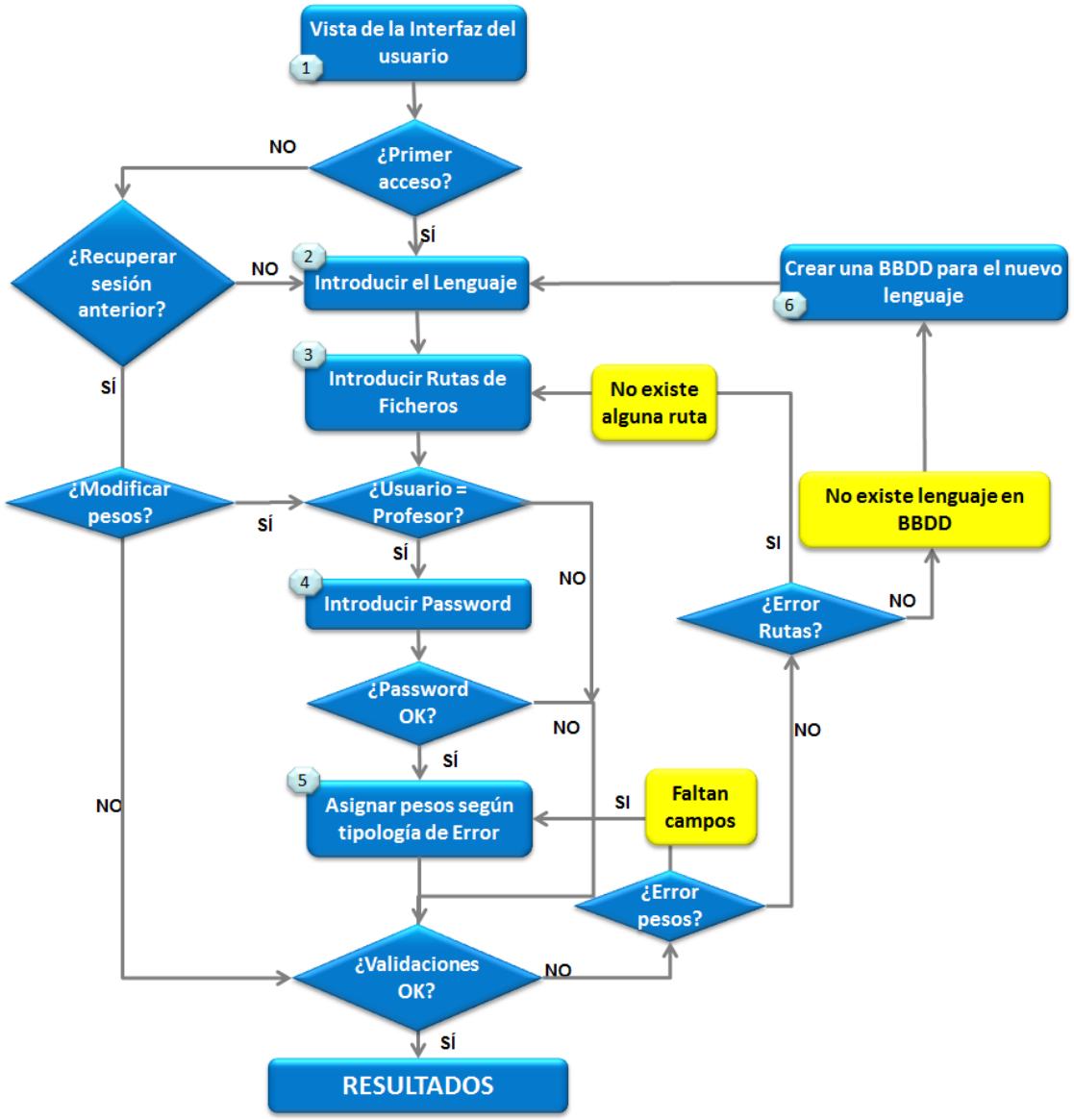


Figura 64. Diagrama de flujo de la herramienta

Se ejecuta el proyecto en Java pulsando RUN

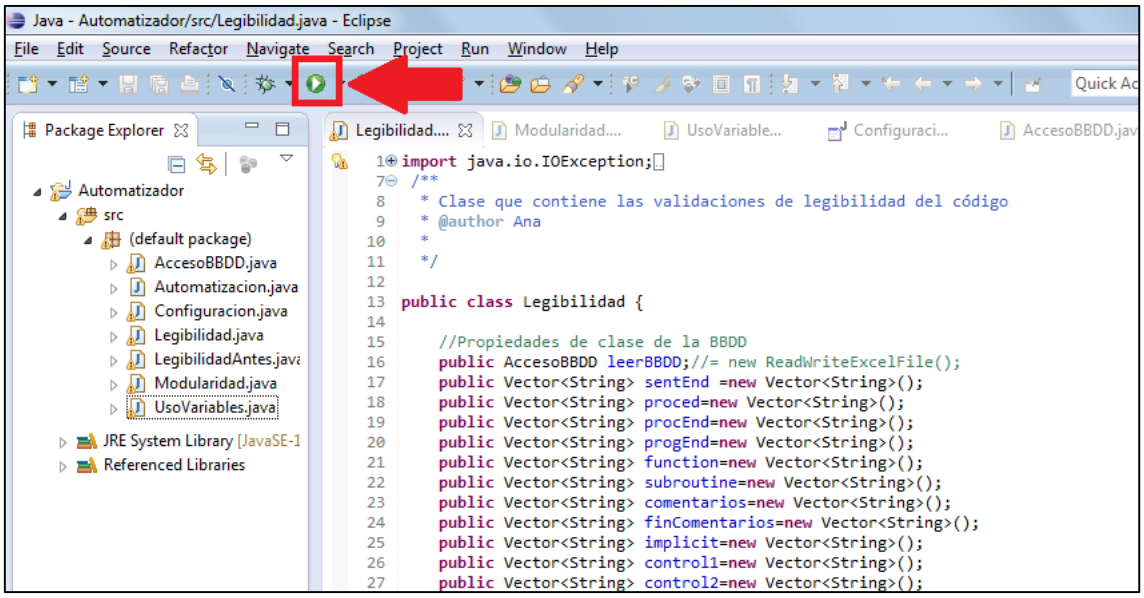


Figura 65. Ejecutar el proyecto en Java

Y a continuación seguimos las ramas del diagrama:

1. *Vista de la Interfaz de usuario*

Tras la ejecución se inicia la herramienta y aparece la interfaz siguiente:

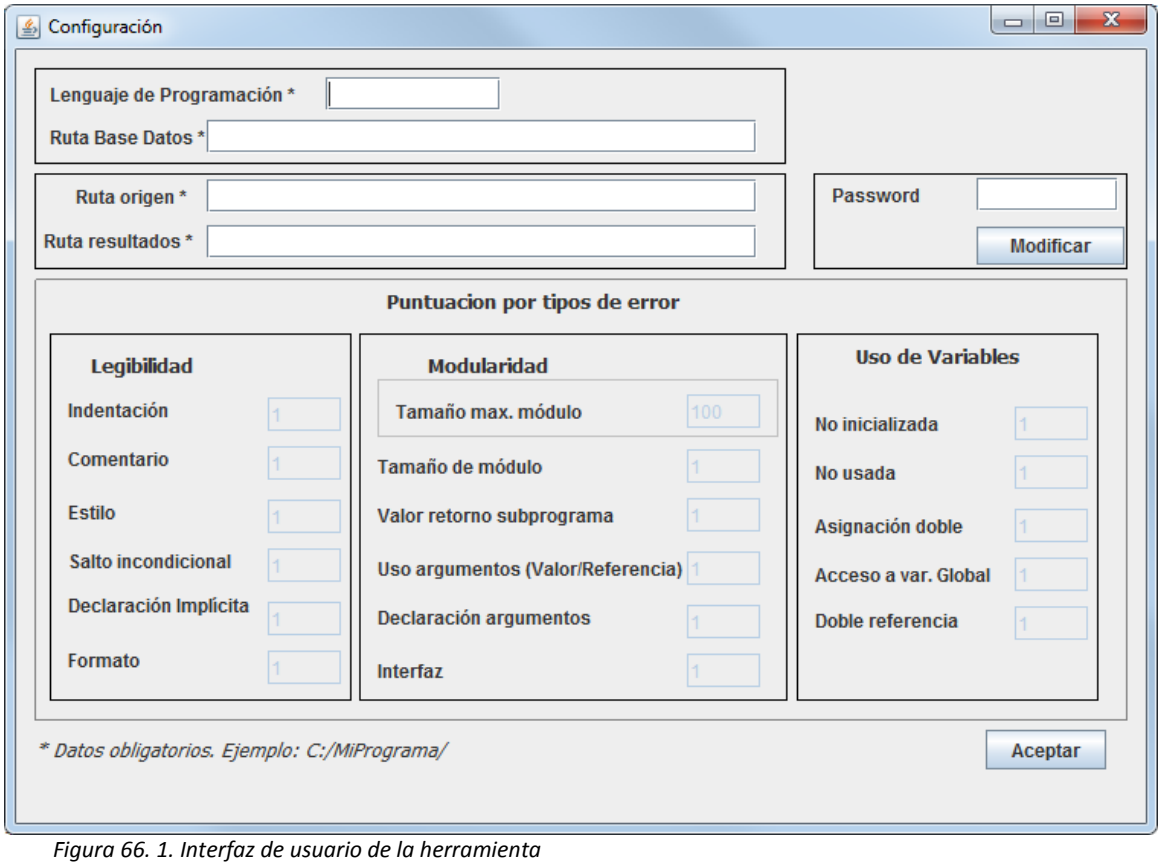


Figura 66. 1. Interfaz de usuario de la herramienta

Si ya se han guardado una sesión anterior, aparece el mensaje para recuperan los datos introducidos:

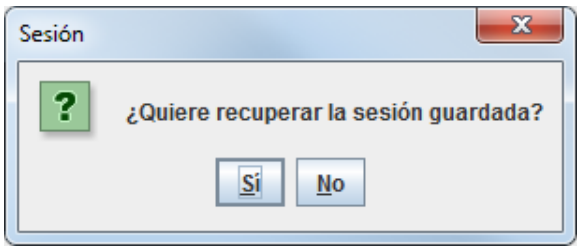


Figura 67. Opción de Recuperación de sesión

Opción Recuperación de sesión: SI (deseo recuperar la sesión anterior)

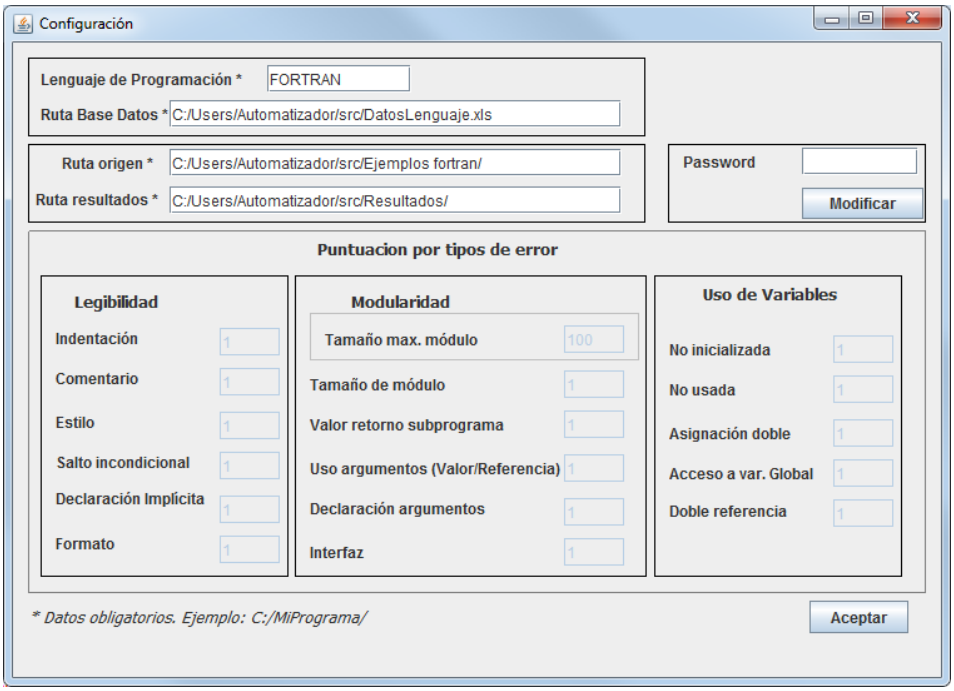


Figura 68. Opción Recuperar sesión = SI

Continua en *4. Introducir Password*

Opción Recuperación de sesión: NO (no deseo recuperar la sesión anterior)

Continua en *2. Introducir Lenguaje.* Equivale a no tener sesión guardada.

2. *Introducir Lenguaje*

El usuario introduce un lenguaje de programación en el campo llamado Lenguaje* de Programación. Continua en *3. Introducir Rutas de Ficheros*

3. *Introducir Rutas de Ficheros*

Se introducen todas las rutas, la del fichero Excel de la base de datos y las rutas donde deseamos que se guarden los ficheros: Resultado y Documentación automática para cada uno de los programas que se desea corregir.

Se completan los campos obligatorios:

- Ruta Base Datos*
- Ruta origen*
- Ruta resultados*

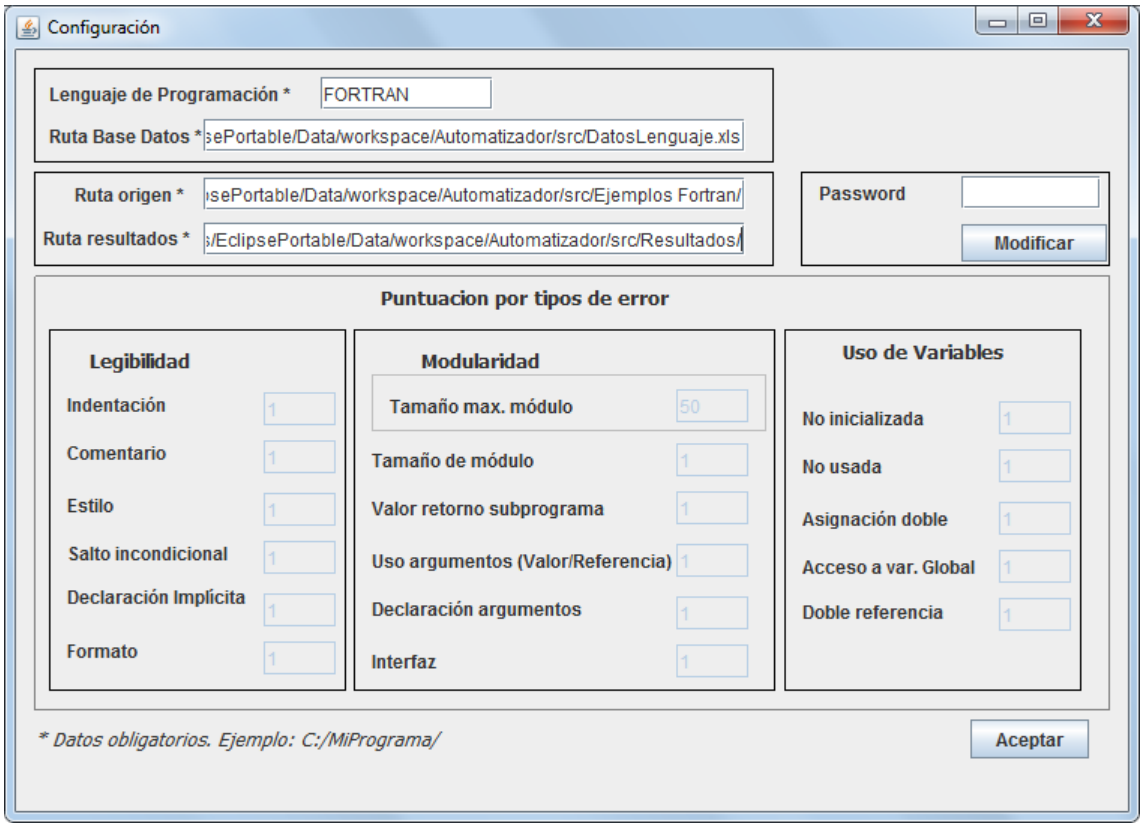


Figura 69. 3. Introducir rutas de ficheros

Continua en *4. Introducir password*

4. Introducir Password

- Si el usuario lo conoce puede introducir el password. Continua en MODIFICAR
- Si el usuario no lo conoce. Continua en *5. Asignar pesos por tipología de error*

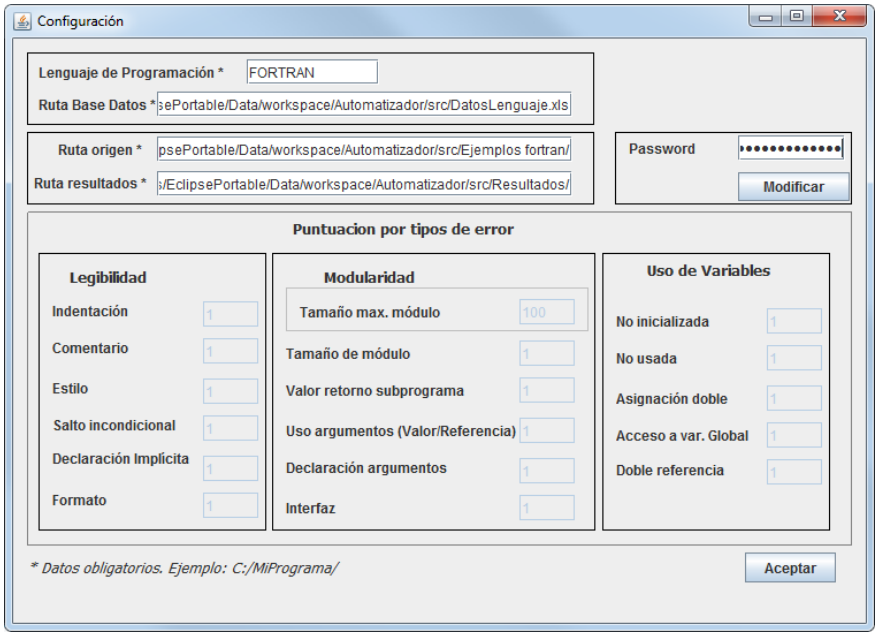


Figura 70. 4. Introducir password

MODIFICAR

Se pulsa el botón Modificar:

- Password correcto (rol Profesor): Continua en *5. Asignar pesos por tipología de error*
- Password incorrecto (rol Alumno): Continua en ACEPTAR

5. Asignar pesos por tipología de error

Se habilitan los campos de la sección Puntuación por tipos de error y se modifican según criterio de evaluación

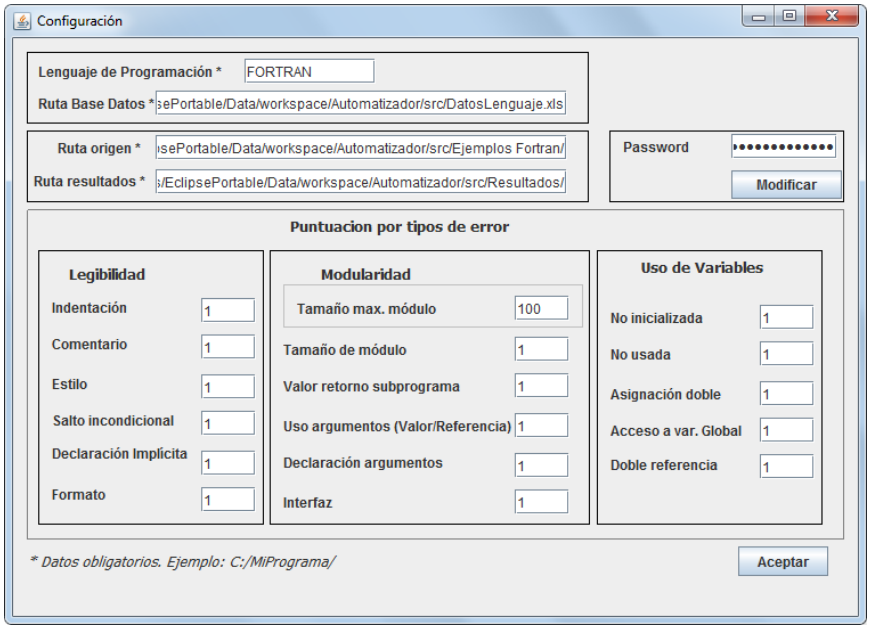


Figura 71. 5. Asignar pesos por tipología de error

Continua en ACEPTAR

ACEPTAR

Se realizan todas las validaciones siguientes:

Validación Pesos: valores vacíos

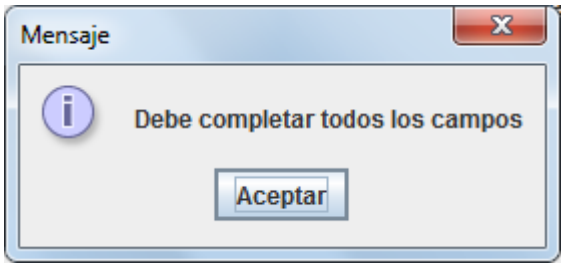


Figura 72. Mensaje de aviso de pesos incompletos

Continua en *5. Asignar pesos por tipología de error*

Validación Fichero: no existe fichero o ruta mal introducida

Esta validación solo se realizara en la ruta de fichero de origen y en la de la ruta de BBDD, en donde se realizan lecturas. En caso de no existir la ruta de resultados, al ser de escritura, se la crearía.

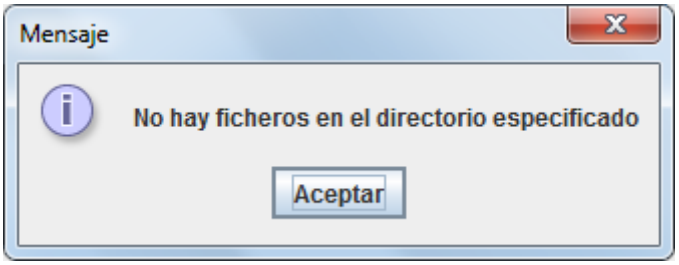


Figura 73. Mensaje de aviso de error en fichero

Continua en *3. Introducir Rutas de Ficheros*

Validación Lenguaje: no existe en BBDD

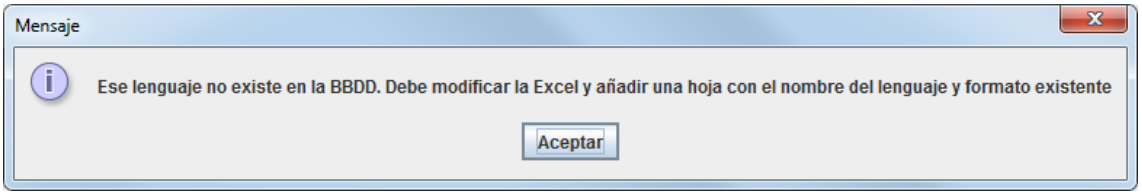


Figura 74. Opción Lenguaje existe en BBDD: NO

Continuo en *6. Crear una base de datos para el nuevo lenguaje*. Debemos crear una nueva base de datos con el lenguaje introducido, para ello accedemos a la Excel facilitada e introducimos los datos en las columnas.

6. Crear una base de datos para el nuevo lenguaje

Para el diseño de la base de datos de un nuevo lenguaje se crea una nueva hoja en la Excel y se crean las mismas columnas en otro lenguaje existente. Se completan de la siguiente manera:

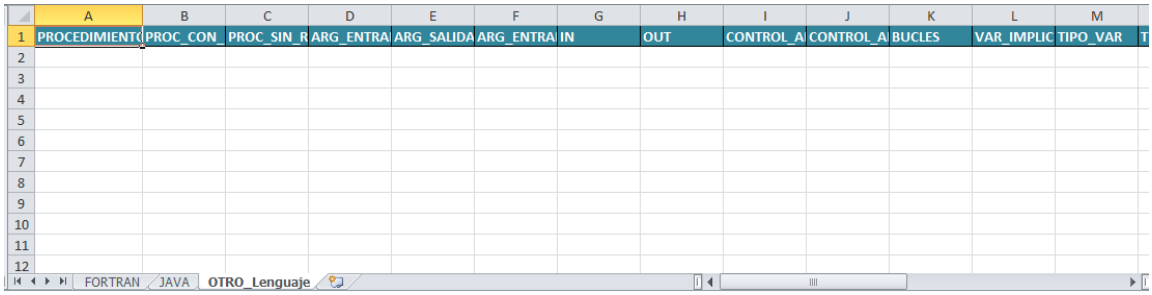


Figura 75. Crear una base de datos para el nuevo lenguaje

- **PROCEDIMIENTO**: Se introduce las palabras restringidas que indican un nuevo procedimiento o programa principal independiente de cualquier otro.
- **PROC_CON_RETORNO**: Subprocedimientos, dentro o fuera del principal que devuelve algún valor.
- **PROC_SIN_RETORNO**: Subprocedimientos dentro del lenguaje que no devuelve ningún valor.
- **ARG_ENTRADA**: Sentencia o palabras reservadas que identifica que los argumentos son pasador por valor, son de entrada y no pueden modificarse durante la ejecución del procedimiento.
- **ARG_SALIDA**: Sentencia o palabras reservadas que identifica que los argumentos son pasados por referencia, son de salida y su valor se modificará durante la ejecución del procedimiento.
- **ARG_ENTRADA_SALIDA**: Sentencia o palabras reservadas que identifica que los argumentos son pasados por referencia, pueden ser de salida y de entrada son modificables.
- **IN**: Inicio de instrucción o sentencia que indica la lectura de un dato por teclado u otro interfaz de entrada
- **OUT**: Inicio de instrucción o sentencia que indica la escritura de datos por pantalla o cualquier interfaz de salida
- **CONTROL_ALTERN_1:** Son las sentencias de control alternativas que implican disminuir la indentación del código
- **CONTROL_ALTERN_2**: Son las sentencias de control alternativas que implican aumentar la indentación del código
- **BUCLES**: Sentencias de control repetitivas
- **VAR_IMPLICITA**: Sentencia que indica si la definición de variables debe ser explícita
- **TIPO_VAR**: Tipos de datos que determinarán el espacio de memoria en el que se almacenan un valor de identificador (pueden ser variables y constantes)
- **TIPO_CONST**: Declaración que define un valor específico y determinado a un determinado identificador.
- **FIN_LINEA**: Un identificador o conjunto de caracteres que indican el final de una línea de código
- **COMENTARIO**: Un identificador o conjunto de caracteres que indican el comienzo

de un comentario, a partir del cual no se considera código compilable

- **FIN_COMENTARIO**: Un identificador o conjunto de caracteres que determinan el final de un comentario.
- **OP_ASIGNACION:** Carácter o conjunto de caracteres que asignan un valor a un identificador (variable o constante)
- **OP_ARITM:** caracteres especiales que contienen las operaciones: suma, resta, multiplicación, división, potencia,…
- **OP_RELAC**: caracteres especiales que contienen las operaciones de: igual que, distinto de, mayor que, mayor o igual que, menor que, menor o igual que.
- **OP_ALFANUM**: caracteres especiales que contienen operandos (caracteres y cadenas) para la concatenación
- **OP_LOG**: caracteres especiales que contienen las operaciones "o ", "y" y "negación" lógicos
- **PARADA**: instrucción o sentencia de para la ejecución del procedimiento
- **SALTO_INCONDICIONAL**: instrucción que indica al procedimiento en curso que debe continuar su ejecución en otra parte del código delegando la responsabilidad sobre aquella parte del programa.
- **FIN_SENTENCIAS**: indicador de fin de sentencias de control alternativas y repetitivas
- **FIN_SUBPROCEDIMIENTOS:** indicador de fin de subprocedimientos o subprogramas
- **FIN_PROCEDIMIENTO:** indicador de fin de procedimiento o programa principal
- **LLAMADA:** instrucción que indica que el procedimiento tiene que continuar su ejecución en otra parte del código (también llamada subrutina) y cuando finaliza la ejecución devuelve el control sobre el procedimiento que la llamó.

Una vez completada la Excel. Continua en *2. Introducir Lenguaje.*